AFIT/GST/ENS/87J

APPLICATION OF ARTIFICIAL INTELLIGENCE IN A CONFLICT ANALYSIS DECISION AID

THESIS

Rollin J Lutz, Jr.
Major, USAF

AFIT/GST/ENS/87J-10

AFIT/GST/ENS/87J

APPLICATION OF ARTIFICIAL INTELLIGENCE IN A CONFLICT ANALYSIS DECISION AID

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Rollin J. Lutz, B.S.
Major, USAF

June 1987

Preface

The purpose of this research was to demonstrate the feasibility of blending operations research and artificial intelligence techniques. A unique prototype system was developed. The prototype, an expert system, was developed and totally integrated, under the control of a C language program.

The success of this research would not have been possible without the great support provided to me by many individuals. To those dedicated people, I am deeply indebted. I thank my advisors, Lt.Col. Parnell and Colonel O'Connell for their professional guidance throughout this work. I also wish to thank Mr. Robert Carrol and the staff at Software A&E for their support in answering my many questions. Without the special version of KES, which they so graciously provided, this effort would not have possible. I wish to express my appreciation to the Operational Sciences Department of the Air Force Institute of Technology for providing the resources which made this effort possible. In particular though, I wish to thank my wife, Shelley and daughter, Marjorie for their love, patience and understanding. They dragged me back to sanity from the perpetual late hours at the computer, in which I was forever engulfed.

Table of Contents

List of Figures

List of Tables

List of Symbols

AFIT- Air Force Institute of Technology
AFB- Air Force Base
AI- Artificial Intelligence
ASE-Advanced Sensor Exploitation
ASM- Assembler Programing Language
AT- Advanced Technology (IBM Computer System)
ATAF-Allied Tactical Air Force

CENTAG
COMNORTHAG-Commander NORTHAG
CONUS- Continental United States
CR- Carriage Return

DIA- Defense Intelligence Agency
DIR- Directory
DOS- Disk Operating System

ECH- Echelon
EGA- Enhanced Graphics Adapter
EOB- Electronic Order of Battle
EXE- Executable File
ES- Expert System

FRG-Federal Republic of Germany

GDP-General Defense Position
GDR-German Democratic Republic

IBM- International Business Machines
IGB- Inter German Border

KB- Knowledge Base
KE- Knowledge Engineering
KES- Knowledge Engineering System

MS DOS- Microsoft Disk Operating System
NATO- North Atlantic Treaty Organization
NL- Netherlands
NORTHAG-Northern Army Group

OBJ- Object File
OMG-Operational Maneuver Group

PC- Personal Computer
PF-Present Force only
PKB- Parsed Knowledge Base File

SACEUR-Supreme Allied Commander, Europe

TVD-Western Theater of Military Operations

UI- Unilateral Improvement

WP- Warsaw Pact

XT- (IBM Computer System)

Abstract

This report documents the research conducted to develop a decision aid for an operations research methodology, conflict analysis. The purpose of this research was to demonstrate the feasibility of blending artificial intelligence and operations research techniques. A prototype system was designed, developed, and implemented in the form of an expert system. Unique in this design, was the expert system embedded completely within a conventional C language program. The prototype automated many of the tedious calculations and processes performed in a classic conflict analysis.

The development process was conducted in three phases. First, the system requirements were established. Second, design tradeoffs were made in the choice of tools and software. Third, the prototype was built using two parallel development tracks. Separate design components were completed in the conventional C language program and in the expert system program. When fully tested and debugged, the two programs were integrated within the C language program. Proper system performance was evaluated by comparison of results using a varied selection of classic problems. Finally, the results of the research are presented with recommendations for future work.

APPLICATION OF ARTIFICIAL INTELLIGENCE IN A CONFLICT ANALYSIS DECISION AID

I. Introduction

Artificial Intelligence has made significant inroads in military and commercial decision aids. Research in this field is growing rapidly as new and improved tools are applied to difficult problems. This research focuses on the design of a computer decision aid by applying expert system tools to the military intelligence analysis of conflict situations.

Topic Statement

This report covers the research and development of an expert system which can aid a military analyst. The purpose of this effort was not to replace the expert, rather to give the analyst a tool. Using such a tool, the process is accelerated and the analyst has time to view the larger picture without being mired in the details. Although not the primary objective, this effort should produce a system friendlier to the user.

Overview of Military Intelligence in Combat Operations

Warfare is both a science and an art as old as the earliest history of mankind. Approximately 2500 years ago Sun Tzu noted, "The art of war is of vital importance

to the state. It is a matter of life and death, a road either to safety or to ruin. Hence, under no circumstances can it be neglected."[14] Sun Tzu's conclusions are still applicable to conflicts in the world today in spite of all the technology changes.

The technology of Chinese weapons in 500 B.C. was admittedly crude by modern standards. Bows and arrows soon gave way to gun powder, but armies still marched over the continent at the same speed. Gun and artillery batteries projected their force in ever increasing distances as technology improved. Once the armored tank appeared in World War I, however, the speed of battle was forever altered. Aircraft soon appeared, projecting power greater distance and more rapidly than ever before. The effect of these advances in weaponry was greater destructive power delivered in decreasing amounts of time. The military commander, however, has changed very little throughout history. He must still assess, decide, and act in much the same manner as Sun Tzu only he must do it in less time.

> Prior to World War II, the effectiveness of a commander depended almost entirely upon his ability to remember and to act upon the tenets of what might well be termed a Commander's Catechism. The pace of war until that time was such that a commander, aided by his staff could gather the necessary information, weigh each factor carefully, generate and evaluate reasonable hypotheses, and identify appropriate option alternatives for command decisions. The speed of human thought and even of organizational interaction, bureaucratic as it may have been, was more or less adequate to the task; and from a communications standpoint the "runner" had already been replaced by the motorcycle rider and by the wire line communications.[19:619]

World War II produced many advances such as the jet aircraft, the German Blitzkrieg, and radio communications. The increased speed of operations stretched the capabilities of military organizations to react. Organizational changes were

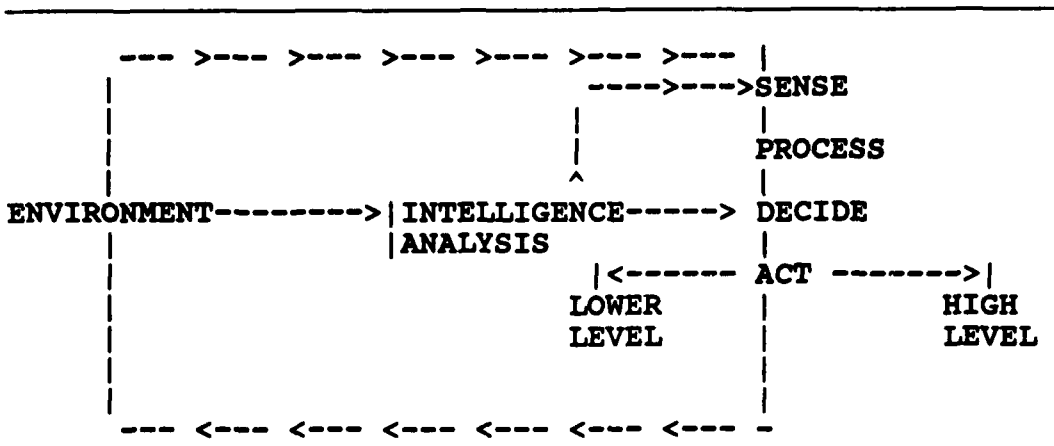
made to streamline the decision making process. The ... of fighting inside the enemy "decision cycle" began to emerge.

The concept that the commander must assess, plan, and act within ever shortening periods of time is important to current operations. Shortening the decision cycle does not mean a less thorough assessment, plan, or action is acceptable. Returning to Sun Tzu's advice on planning:

> The general who wins a battle makes many calculations in his temple before the battle is fought. The general who loses a battle makes but a few calculations beforehand. Thus do many calculations lead to victory, and few calculations to defeat; how much more no calculation at all! It is by attention to this point that I can foresee who is likely to win or lose. [14]

The quality of the assessment and plan must not be sacrificed for the sake of expediency. The detail, timeliness, and quality of the assessment are products of the intelligence analysis. One conceptual model presented by Major Orr in a recent paper, explicitly includes the intelligence function (see in figure 1). The intelligence function is always present in the command and control structure, but plays a diminished role at the lower levels. The output of the intelligence analysis feeds the decision making process of the command structure and aids in the formulation of strategy.

Figure 1. Conceptual Combat Operations Process Model
[8:27]



The intelligence analysis as a support function in the decision making process has become so important that it now exists among government agencies whose functions are to provide necessary analysis. The Central Intelligence Agency, for example, created by congressional charter, provides the government with wide ranging views of the foreign affairs of other governments. The Defense Intelligence Agency, as another example, provides analysis for the Department of Defense oriented toward military affairs. The analysis of conflict situations is one function of such intelligence agencies and is the focus of this research.

The Future of Intelligence and Decision Aids

The explosion of information technology in the 20th century has compounded the military commander's problem of getting inside the enemy's decision cycle. The variety of sensors ranging from radar to chemicals has extended the human senses and taxed the analyst who must sort the noise from the intelligence. The military planner is often overwhelmed by the speed at which

events change and for which operations must be measured and carefully evaluated in the light of increased knowledge the assessment provides. The challenge is to employ the technology of the 20th and 21st century to help the commander make better decisions in less time.

Looking at the technologies of the next two decades, Air Force Systems Command conducted Project Forecast II to highlight the most promising new research areas. Gen. Skantze has described one outgrowth of the study as the "super battle management station"[11:32]. The battle management station would use Artificial intelligence (AI) based decision support, make high speed simulation for planning "what if" exercises and communicate in a highly visual representation. With such a battle management station, the commander could greatly decrease decision cycle time and still make quantum improvements in the quality of his assessment and planning.

One of the key technologies for developing the battle station is AI. It has received rapid recognition as the leading technology and has enabled development of the "expert system". Expert systems, used in decision support systems, can assist judgment or choice. One example of such a system (designed along the lines that General Skantze outlined) is the Advanced Sensor Exploitation (ASE) at Rome Air Development Center [13:105]. Graphics are employed to display high resolution radar and strike locator data in a simulated battle management center. AI is used by the analyst within certain bounded problem domains to make inferences such as the message traffic flow and volume identifying the nature of a given location. The use of AI languages in lieu of conventional languages lends many advantages. The systems can be rapidly prototyped, the knowledge base is formed in natural, language-like rules, and allows for operations with uncertain or incomplete data.

Another valuable feature of AI based systems is that the logical chain of decision rules can be followed or displayed upon query to show recommendation logic.

A second concept flowing from Gen. Skantze's battle station is the real time 'what if' simulations. Most of the operational planning follows from the 'what if' to develop options. Application of a game theory model in this battle station would seem to be a natural since the basis is a conflict of two or more players. One application of game theory ideas to real world conflicts is the development of conflict analysis by Fraser and Hipel [3]. The most appealing aspect of the conflict analysis method is that it is nonquantitative in its modeling of sociological and psychological properties inherent in conflict situations. The relative preferences of the players are developed and resolved by mathematical set theory and logic.

Artificial Intelligence

Following World War II computer scientists began an intensive effort to make electronic machines think, act, and behave like humans. Although it seems the computer has been around for a long time, it is a relatively recent product. British and American scientists in the 1940's worked in separate groups to bring forth the electronic computer. In the early design, Alan Turing, a British scientist, proposed that the controlling instructions fed to a computer should be based on logical operators such as "or", "not", or "and" [4:2]. The advantage of such a system would be the capability to manipulate symbolic information such as natural language. American designers, however, chose to pursue a machine based on numerical operators such as "+", "-", and ">". The American design became the defacto standard. The growth of technology to the current crop of super computers has focused along the numerical processing. However, recent research has revisited

the early design choices and brought forth the interdisciplinary approach to computer science called Artificial Intelligence (AI).

In the the 1960's AI began to emerge from universities. Predictions of humanoid butlers to do household chores, machine translators that could speak any language, or robots that would do dirty or dangerous jobs abounded [10:195]. Belief that machines might possess such human qualities gave rise to concerns that machines might become smarter than humans. While computer science is developing at an exponential rate, the tasks of human decision-making, robotics, and speech have proven to be more troublesome than first envisioned. While a total solution has not been achieved, methodical progress in specific aspects of AI problems has proven productive. One of the most recent advances is the development of expert system tools which allow the design of computer decision making or diagnostic programs for very specific applications.

Expert Systems

From the outset, AI has worked to bring human behavior to the computer. The primary focus of artificial intelligence is abstract problem solving. In contrast, the "expert system" or "knowledge system" differs from other AI techniques in that knowledge is gathered and applied to a specific event or problem just as a human expert would do. The term "expert" can be widely applied, but a working definition is given by Johnson, a scientist who studies the decision making of human experts:

> An expert is a person who, because of training and experience, is able to do things the rest of us cannot; experts are not only proficient but also smooth and efficient in the actions they take. Experts know a great many things and have tricks and caveats for applying what they know to problems and tasks; they are also good at plowing through irrelevant information in order to get at basic issues, and they are good at recognizing

> problems they face as instances of types with which they are familiar. Underlying the behavior of experts is the body of operative knowledge we have termed expertise. It is reasonable to suppose, therefore, that experts are the ones to ask when we wish to represent the expertise that makes their behavior possible. [17:5]

Professor Edward Feigenbaum, a noted researcher from Standford University has defined an expert system as:

> ... an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. Knowledge necessary to perform at such a level, plus the inference procedures used can be thought of as a model of the expertise of the best practitioners of the field.
>
> The knowledge of an expert system consists of facts and heuristics. The "facts" constitute a body of information that is widely shared, publicly available, and generally agreed upon by experts in a field. The "heuristics" are mostly private, little discussed rules of good judgment (rules of plausible reasoning, rules of good guessing) that characterize expert-level decision making in the field. The performance level of the expert system is primarily a function of the size and the quality of a knowledge base it possesses. [1:5]

The modeling of the human expert has provided the computer the capability to perform difficult decision-making. Ideally, the expert system functions just as the human experts commonly do. For example, experts ask questions, apply prior general knowledge, and make inferences. Often, the problem presents the expert with uncertain or incomplete information that must be accounted for in solving the problem.

Research Problem

The specific research problem is to assist the intelligence analyst by automating parts of a conflict analysis. Expert system technology will be explored to

test the feasibility of capturing enough knowledge of conflict analysis to develop and implement a personal computer aid.

Objective

The objective of this research is to demonstrate the feasibility of blending AI and operations research techniques by applying expert system technology to conflict analysis. Inherent in this approach is the development of a working expert system model that can run on an IBM PC or similar system. In pursuing the design and development of this system, limitations of the use of small computer environments as aids should be understood. An evaluation of the system should also provide insights into future benefits to be derived.

Scope

The average intelligence analyst brings a broad base of perceptual tools to bear on any one problem that is far beyond the scope of this research effort. The problem domain is narrowed to view only events which may be modeled as conflict situations. While other tools are available for assessing outcomes, this work is focused solely on resolutions of small conflict models. The number of players is limited to two, and each player may form 5 options. These limits may be readily expanded in future extensions but serve to demonstrate the feasibility of the approach. This research encodes the knowledge base and heuristics of a conflict analysis. The stability output was modeled as described by Hipel and Fraser [3:14] and displayed in a familiar form to the analyst.

Assumptions

This research presupposes that the user of this system is not unfamiliar with the techniques of conflict analysis. The user must provide the system with appropriate information. It is not designed as a replacement for the skilled analyst. Instead, it is meant to aid or advise. Computer literacy is also assumed, a minimum of normal startup and running of programs under the MS DOS. The operator should be able to respond to the normal DOS commands and be comfortable with its operation. The government analyst frequently must assess various security classifications of the information and its subsequent products. This research does not attempt to assess the security aspects of the work. This is left to the analyst.

Equipment

The equipment needed to support this research maybe cast into two categories; computer hardware and computer software. The computer hardware consists of the electronic central processor unit (CPU) and its attendant support equipment; video monitor, input/output devices, mass storage, and power supplies. The hardware system which this computer aid supports is the IBM Personal Computer or a close compatible. Although the target system is the IBM PC, some of the development was conducted with the aid of a VAX 11/785, Zenith Z-248 PC., and an IBM Advanced Technology (AT) PC. Since the software needs specific hardware configurations in order to function correctly, a minimum system will require 512k bytes of random access memory (RAM), a 5.25" floppy drive, a fixed drive with at least 2M bytes, and a graphics monitor. The development system was connected to the VAX 11/780 via a serial modem and required serial interfaces not needed by the target systems. The software environment was run under the

Microsoft Disk Operating System (MS-DOS) version 3.0, 3.1, and 3.2. Primary system development was written in Microsoft C version 4.0 and integrated with Knowledge Engineering Systems (KES) version 2.3.

Overview

The remaining chapters of this research report describe specific portions of the development. Chapter II will introduce the reader to the methodology used in conflict analysis. Chapter III details the methodology used. Chapter IV summarizes current literature pertaining to expert systems or knowledge based systems in the military and government. Chapter V describes the expert system design for this research. This chapter also develops the conceptual model for the overall system and examines the design tradeoffs. Chapter VI describes the working prototype by guiding the reader through a small analysis using the system. Finally, Chapter VII states the research conclusions, future development extensions, and thesis recommendations.

II. Conflict Analysis

Introduction

The central issue in this research is the application of AI to an operations research technique, conflict analysis [3:6]. This chapter presents an example of the conflict analysis methodology. The steps in the analysis are explained using a hypothetical NATO scenario. Many of the terms, as well as the overall process presented here, are applied in later chapters during the development.

Conflict is the opposition of people or forces and is ever present in the real world around us. It is present in small claims court, labor relations, business, sports, politics, and in the most extreme case we have war. A systematic study of conflict can be extremely useful, especially if the outcome can be forecast in advance. One field of study that has concentrated on this area of conflict is game theory. Using game theory as one of his many tools, an intelligence analyst can structure and ideally estimate possible resolutions or outcomes.

Classical Game Theory

Game theory had its beginning in the theoretical work of Von Neumann. Von Neumann and Morgenstern published their book the Theory of Games and Economic Behavior in 1944 [16] and thus established what has commonly been termed classical game theory.

Much of classical game theory builds upon the two-person zero-sum game. Two adversaries, called players, compete in a game, and as one wins the other loses

a like amount. Since the sum of the net winnings and losses is zero it is a zero-sum game. Each game by definition involves some rules which force the players to take certain actions or suffer some consequence. The players then form their winning strategies to achieve their goal. The goal may be to maximize the winnings, or to minimize the losses. Whatever the goal, the strategy is chosen to achieve the desired effect with the payoffs. The implicit assumption here is that the game has value and can be modeled using payoffs. Such payoffs can be measured in dollars, resources, utils, or any compatible unit of measure between the players. A strategy is then determined by the cardinal values associated with the payoffs. Greater or lesser value achieved by strategy number one is compared to strategy two, etc. until all the options are evaluated. The rational player then will pick the strategy or combination of strategies to maximize or minimize the payoff.

To illustrate the two person zero sum game consider the following game given by Hillier and Lieberman.

> To illustrate the basic characteristics of a game theory model, consider a simplified version of the game called two fingered morra. In this version, each player simultaneously shows either one or two fingers. If the number of fingers match, then player I wins, say, $1 from player II. If they do not match, player I would pay $1 to player II. Thus each player has two strategies: to show either one or two fingers.[5:300]

The strategies 1 and 2 are presented in the payoff table in Figure 2.1. If player Player I shows one finger, then player II would like to have two. If player I shows two fingers, then player II would like show only one since this would get him the best result. Without prior knowledge of the other players strategy, however, there is no clear best strategy. The choice of strategy in this example is determined on the cardinal values each achieves as shown in the payoff table.

Table II-1 Payoff Table for Two Finger Morra

		Player II 1	2
Player I	1	1	-1
	2	-1	1

MetaGame Theory

From the preceding example of a game, it is clear that the formation of the win-loss or value table is needed. Often the values for such a table in real life may not be easily established. The unit of measure often used is the dollar, and that can be misleading. How much for example, is a human life in combat worth - one tank, a battalion of artillery, or maybe a single hill. Placing values on such things is exceedingly difficult. One method to evaluating games without such quantitative values, is to use mathematical set theory and form relationships which will allow us to solve the problem. In 1971 Howard publish his book [6] on metagame theory which established such a methodology. Additional work by Fraser and Hipel [1] established an extended metagame theory, conflict analysis. Conflict analysis is the method used in this research.

Modeling a Hypothetical NATO Conflict

Perhaps the best way to understand the use of conflict analysis is to use an example and to explore its development. The example chosen for this task is taken from a research paper by Capt John Yanaros, an AFIT graduate student. While

many analyses are done in hindsight, Capt Yanaros' example presents a 'real world' look at a hypothetical future NATO conflict. It is presented here for demonstration of the conflict analysis technique, not as a advocate of any particular point of view of the military or political events in NATO. The conflict situation in NATO is widely studied, providing grounds for many of the assumptions and allowing an unclassified body of information about the problem.

Define the Conflict

The first task in the analysis is to define the conflict. The players, each players options, and general background information pertinent to the structure of the opposing players are established. After the initial setup; the various outcomes are generated, unlikely outcomes removed, preferences ordered, and finally a stability analysis provides possible equilibriums. The following hypothetical scenario introduces the players and the game structure which will be used throughout this report as an example.

Hypothetical Scenario

> U.S. involvement in a Central American conflict, instability in the Middle East, removal of US and Soviet tactical nuclear weapons following an Arms Reduction Treaty, and perceived weakness in NATO unity has incited the Soviets to attack Western Europe, the Baltic countries and the Northeastern Meditteranean. Prior to this move, however, increased tensions signal the Allies. The anticipation of possible overt hostilities of the Warsaw Pact (WP) on NATO territories calls for mobilization and a current analysis of the friendly and enemy order of battle (FOB/EOB) to verify current war plan applicability and identify possible attack avenues of approach to shift current peacetime general defense positions (GDP) -- if necessary -- of NATO forces. Full mobilization is not accomplished, however, due to WP deception through extended exercises over the past few months. Analysis is done for the NORTHAG commander (COMNORTHAG) to provide him with expected WP actions and possible results of the initial hours of the conflict. [20:4]

Players

> ...only the NORTHAG/2ATAF area is addressed in the analysis. Therefore, the NATO player is COMNORTHAG -- referred to as the NATO player -- and as mentioned ... WP will refer to the second player....[5:5]

Once the players are established, they can be represented in a variety of ways. One method that is used in this report, is to place the players and their options in a table format that can be used to represent the conflict. Table II-2 shows the beginning of this tabular formation with the players each positioned on the far left. The options for each player will follow under the respective player.

Table II-2. Game structure -- the players

PLAYERS	PREFERENCES VECTORs
I. Warsaw Pact	
II. COMNORTHAG (NATO)	

The objectives and an understanding of the overall goals by a player are used in the ranking of preferences. The objectives also clarify some the players options. For this analysis the players options and objectives are as follows:

NATO/WP Objectives

> Actions by the players in the ground conflict are influenced by their respective national and warfighting objectives and corresponding doctrine. Objectives are...
>
> 1. Maintain present IGB
>   a. mobilize reinforcements quickly
> 2. Slow WP advance (decrease momentum)

3. Counterattack to defeat WP
4. Restore pre-war IGB
5. Avoid a nuclear conflict if possible

WP overall objective is a reflection of USSR Foreign Policy -- ... For the NORTHAG region, the Western TVD wishes to:

1. Secure primary northern FRG power source at Hamburg (Nuclear Power Facility on the Elbe River)

2. Secure Bremerhaven (a port for NATO reinforcements)

3. Flank to the South into West German and British corps sectors to meet central WP forces at Koln.[5:6]

WP Options

The general options are to attack the NL Corps sector with:

1. 1 echelon
2. 1 echelon with an OMG
3. 2 echelon
4. 2 echelon with an OMG
5. OMG only [5:17]

1 echelon (abbreviated 1 ECH)
2 echelon (abbreviated 2 ECH)
Operational Maneuver Group (abbreviated OMG)

With the Warsaw Pact options established, we return to the table format and place the options for the WP player under his position in the table (see Table II-3 for the format). Since the NATO options are yet to be entered, we continue with the NATO objectives in the scenario.

Table II-3. Game structure -- the players and options

PLAYERS	PREFERENCES VECTORS
I. Warsaw Pact	
1. 1 ECH	
2. 1 ECH/ OMG	
3. 2 ECH	
4. 2 ECH/ OMG	
5. OMG	
II. NATO	

NATO Options

...it is assumed that the German Territorial Army assets assigned to the NL Corps sector will defend there. NATO Naval air/amphibious options are not considered...

The COMNORTHAG options are:

1. Use of present (Netherland heavy armored brigade) force only. Wait for NL reserves that are five hours from the peacetime GDP.

2. Move only Danish/german assets from the BALTAP (with SACEUR's approval) -- referred to hereafter as DA or Danish (GErman assets from the Schleswig-Holstien area implied in the notation).

3. Move only German Corps assets (GE) into the NL sector. Allow the British Corps to backup the German Corps if needed.

4. Use the US III Corps Brigade (forward) only. Peacetime GDP is located within the NL Corps forward area.[:18]

Present Force Only (abbreviated PF)
Move DA BALTAP assets (abbreviated DA)
Move GE Corps assets (abbreviated GE)
Use US III Corps Brigade (abbreviated US) [5:19]

Table II-4 now has all the players and their respective options. All information necessary for the representation of the outcomes has been defined. No information, however, has been provided about the feasibility of specific outcomes, or about the players preferences. In order to complete the initial table, the options must be uniquely represented.

Table II-4. Game structure -- the players and options

PLAYERS	PREFERENCES VECTORS
I. Warsaw Pact	
1. 1 ECH	
2. 1 ECH/ OMG	
3. 2 ECH	
4. 2 ECH/ OMG	
5. OMG	
II. NATO	
1. PF	
2. DA	
3. GE	
4. US	

Vector Representation

Once all the players and the options are defined, a binary representation of the possible outcomes is used. When an option is taken, we assign it a value of 1, and when not taken a value of '0'. Thus for the NATO player

II. NATO

1. PF 1

2. DA 0

3. GE 0

the outcome [0001] means that the present force only is used (PF). Notice that the low order 1 is at the top of the list, and the higher order 0 is at the bottom. One additional point in the WP option list, is the way that five options can be represented with just three. From Table 6, for example, an echelon with an OMG is [101] and an echelon by itself is [001]. It is often more convenient to work with decimal numbers, converting the binary number into a decimal equivalent. One method for this, is to form the equivalent base number powers. The binary number 10100 for example, is represented as:

$1 \times (2^4) + 0 \times (2^3) + 1 \times (2^2) + 0 \times (2^1) + 0 \times (2^0) =$

$16 + 0 + 4 + 0 + 0 = 20$ (in base 10).

To reverse the process, we apply Horner's rule [21:9]. Successively, divide the old number by the new base number (10 in this case) and note the remainder at each division. The number 20 in base 10 can be converted as follows:

```
20 / 2 remainder = 0-------------|
10 / 2 remainder = 0------------| |
 5 / 2 remainder = 1----------| | |
 2 / 2 remainder = 0-------| | | |
 1 / 2 remainder = 1-----| | | | |
 0   Binary number =     1 0 1 0 0
```

The use of binary numbers simplifies the problem structure in a compact form. Because each option can either be selected or rejected, in a game involving N options there are 2^N possible outcomes (see Table II-5).

Table II-5. Binary Vector Outcomes Possible

options	possible
1	2 (0 or 1)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

In the case of our NATO example there are 7 total options and thus, 128 possible binary representations. The problem form (see Table 6), now lists all possible outcomes sequentially ordered , not preferentially. It is easy to see how quickly a problem with just two or three players and each with two or three options can quickly become very large. In performing this analysis by hand, each of the outcomes must be represented on paper and then analyzed. This is one area where a computer assisted program can be a real time saver. The computer can also very quickly search and sort the outcomes, the next step following outcome removal.

Table II-6. Game structure -- the players, options, and vectors

PLAYERS	PREFERENCES VECTORS
II. Warsaw Pact	
1. 1 ECH	0 0 0 0 0 0
3. 2 ECH	0 0 0 0 0 0
5. OMG	0 0 0 0 0 0
I. NATO	
1. PF	1 0 1 0 1 0
2. DA	0 1 1 0 0 1
3. GE	0 0 0 1 1 1
4. US	0 0 0 0 0 0

Outcome Removal

Outcome removal is the next major step in the solution process. With 128 outcomes mathematically possible, not all outcomes in the set are realistically feasible. It is highly desirable to reduce the set of outcomes to a smaller group, which makes comparing them an easier task. The form for representing a removal, is a combination of 0's, 1's, and '-'. The '1'and '0' are in the relative positions that are to be matched, and all other don't-care positions are indicated with a '-'. Infeasible outcomes generally are of four types:

Type 1: Options that are not rationally possible for a single player or that may be mutually exclusive. Type 1 removals for this problem are:

A. (1,1,-;-,-,-) WP mutually exclusive

B. (0,0,0;-,-,-,-) Scenario: WP will attack

Type 2: An option which in the opinion of the analyst the players would never be expected to accept. Type 2 removals in this case are:

A. (-,-,-;0,0,0,0)	COMNORTHAG is committed to defend the NL Corps sector with some forces for political reasons and to prevent WP assured flanking success.
B. (-,-,-;0,-,-,-)	The NL Corps brigade is already at the GDP during peacetime; it is reasonable to assume that they are in the sector on D-day.
C. (0,0,1;-,-,-,-)	Soviet doctrine calls for OMG use in conjunction with or in lieu of a second echelon and will not be used if significant opposition in the rear is likely.

Type 3: Options that are not rational between players can be removed. In the NATO example, there are no removals of this type.

Type 4: The last and most judgmental, is the removal of an option which is not feasible for at least one player and which may or may not be feasible for the other player. There are two removals of this type in the example for analysis.

A. (1,0,1;1,0,0,0)	A WP 1ECH/OMG attack would overwhelm NL only GDP.
B. (-,-,-;1,1,1,1)	.....Any combination of 3 NATO forces is sufficient to delay initial invasion. [20:20]

After comparing each outcome with the above removals, there are 33 remaining outcomes in the set. The problem has been reduced from 128 possible outcomes to 33. The remaining outcomes do not reflect any player's preferences, since they are not yet ordered. A preference ordering is the next step, performed for each player separately. The NATO and Warsaw pact preferences are as follows:

> ...The WP prefers to use one or two echelon attacks over those attacks using OMG. Considering the NL Corps terrain (marsh, rolling to flat, and the Elbe River) and surprise ... -- a second echelon attack is preferred over a single echelon attack. The two echelon attack provides mobility and speed into the perceived weak NL Corps area. In summary, the WP attack option preference is:

(0,1,0;-,-,-,-)
(0,1,1;-,-,-,-)
(1,0,0;-,-,-,-).

NATO Preference Vector

COMNORTHAG prefers the following priority of force selection for defense of the NL Corps sector:

1. PF-- the NL brigade is already in position. Prepared positions are easier to defend than hastily prepared ones. No reshuffling means other sectors are more prepared for WP attacks in their areas of responsibility.

2. US--SACUER expressed desire to keep reserve US Corps for CENTAG-- but COMNORTHAG has been given the go ahead to use the US Brigade if he deems it in the best interest of NORTHAG. This force is already forward and can be in position quicker than the GE or DA assets.

3. DA--NATO commanders perceive the BALTAP region under a lesser threat the NORTHAG northern flank. Should a surprise attack or shift of WP assets to the BALTAP occur, SACUER has strategic assets to use in that area, such as naval and amphibious assault forces.

4. GE--COMNORTHAG perceives a heavy WP assault will occur on the I GE corps sector and prefers to keep assets there.

Obviously NATO would prefer a weak versus strong frontal attack and would prefer not to fight an OMG in a weak rear NL corps area. US assumes Soviet doctrine holds and perceives a two echelon attack only. Therefore, the NATO player prefers the following WP attack options, in decreasing order of preference:

1 ECH (1,0,0;-,-,-,-)
2 ECH (0,1,0;-,-,-,-)
1 ECH/OMG (1,0,1;-,-,-,-)
2 ECH/OMG (0,1,1;-,-,-,-)

Knowing the preferences of each player, the set of remaining outcomes is ordered for each player separately. The individual sets of ranked outcomes are the preference vectors for each player. During the ranking process, the outcomes are usually transformed from the binary to the decimal form for ease of display (see

Table II-7). The outcomes at the left of the display are the most preferred and the right most outcomes are the least preferred.

Table II-7 Preference Ordered Outcomes

WP	10	26	74	42	90	58	106	14	30	78
NATO	73	25	41	89	105	57	10	74	26	42

Outcomes Table continued

WP	46	94	62	110	25	73	41	89	57	105
NATO	90	106	58	93	109	61	77	29	45	94

Outcomes Table continued

WP	29	77	45	93	61	109
NATO	110	62	14	78	30	46

Stability Analysis

Once the preference ordering for each player has been determined, the stability of each outcome can be analyzed. The stability process starts by investigating possible unilateral improvements. The subsequent steps in the analysis use the unilateral improvements (UI), to test for possible equilibriums. Once again, this process is highly repetitive and time consuming. Computer searches can greatly speed this process.

Unilateral improvements are improvements in a player's outcome resulting from that player's change of option. For example, the outcomes 73 and 25 for the NATO player:

NATO Preference

```
WP
      1ECH 1 1
      2ECH 0 0
      OMG  0 0

NATO
      PF   1 1
      DA   0 1
      GE   0 0
      US   1 0
====================
           73 25
```

The NATO player, shown on the bottom half of the example, can change the DA option from 1 to 0 and at the same time US from 0 to 1. In making this change the NATO player changes unilaterally and improves from outcome 25 to the next most left outcome, 73. Since the change moved his position to the left or in the more preferred direction, the change is a unilateral improvement. This UI is then written vertically under its corresponding outcome:

```
NATO-73    25
           73
```

Any outcome which has no UIs is labeled rational and a "r" is written above it. Rational, in this case, means that the player would not rationally have a way to change from the current position. To view the all of the binary outcomes for the NATO and WP, see Table 8 .

Table II-8. Basis for UI Calculations for NATO

WP	
1ECH	111111000000011111110000000
2ECH	000000111111100000001111111
OMG	000000000000001111111111111
NATO	
PF	111111111111111111111111111
DA	010101001010110101010010010
GE	001011000101101100101100001
US	100110010011011010011001000

Table II-9. Basis for UI Calculation for WP

WP	
1ECH	000000000000001111111111111
2ECH	111111111111110000000000000
OMG	000000011111110000000111111
NATO	
PF	111111111111111111111111111
DA	010011001001101001101001100
GE	000101100010110010110010110
US	001010100101010101010101010

At this point in the analysis, all outcomes with UI's have been found and the outcomes without UI's labeled as 'r'. The next step determines if a player is stable in a given position. Unstable means there is a way for the player to improve his position-- a UI for which the the opponent can not stop or deter the improvement. When an unstable condition is found a 'u' is written above the outcome. If after checking all the UI's for the preference vector, the player is not able to improve because of the opponents options to sanction his improvement then an 's' is written. The 's' means sequentially sanctioned. For example, NATO's outcome 25 , there is a UI to 73. Checking WP's outcome 25, there is a UI to 26, which for the NATO player is less preferred. Since there is no way to logically improve his outcome,

NATO would be sanctioned by WP and remain at 25. NATO's vector 25 is therefore, sanctioned, 's'. Tables II-10 and II-11 have the completed analysis. All of the outcomes have been resolved to either 'r', 's', or 'u' from which the equilibriums can be found. The equilibrium which are marked with an 'E' are where either 'r' or 's' exist for both players at a given option. The outcome 10, for example, is 'r' for NATO and 'r' for WP making the outcome 10 a possible equilibrium point.

Finding the equilibria is the end of the process, however, it is good practice to revisit many of the earlier assumptions. Outcomes are checked for sensitivity. Performing this work by hand can be very tedious, and the use a computer is recommended.

Table II-10. Stability Table for NATO

NATO

x	x	x	x	x	x	E	x	x	x	x	x	x	x	x	x	x	x	x
r	s	s	s	s	s	r	u	u	u	u	u	u	r	u	u	u	u	
73	25	41	89	105	57	10	74	26	42	90	106	58	93	109	61	77	29	
	73	73	73	73	73		10	10	10	10	10	10		93	93	93	93	
		25	25	25	25			74	74	74	74	74			109	109	109	
			41	41	41				26	26	26	26				61	61	
				89	89					42	42	42					77	
					105						90	90						
												106						

NATO continued

x	x	x	x	x	x	x	x
s	r	u	u	S	u	u	u
45	94	110	62	14	78	30	46
93		94	94	94	94	94	94
109			110	110	110	110	110
61				62	62	62	62
77					14	14	14
29						78	78
							30

Table II-11 Stability Table for WP

WP																				
	r	r	r	r	r	r	r	u	u	u	u	u	u	u	u	u	u	u	u	u
	10	26	74	42	90	58	106	14	30	78	46	94	62	110	25	73	41	89	57
								10	26	74	42	90	58	106	26	74	42	90	58
															30	78	46	94	62

WP continued							
	u	u	u	u	u	u	u
	105	29	77	45	93	61	109
	106	26	74	42	90	58	106
	110	30	78	46	94	62	110
		25	73	41	89	57	105

Summary

Conflicts can be modeled in various ways but game theory has concentrated in this field since Von Neumann started it. Conflict analysis is an extension of game theory and uses ordinal preferences instead of the value modeling of classical game theory. The conflict analysis method has five stages by which a solution is sought. First, the players and options are identified. Once the options are defined, a set of possible outcomes is generated. Third, the infeasible outcomes are removed from the set of possible outcomes. Next, the outcomes are rank ordered by preference. Finally, the stability is analyzed and possible equilibrium outcomes are identified.

III. Methodology

Introduction

This chapter is a guide to the methodology used in this thesis. It is not a detailed design guide, that is left for Chapter V. This chapter concentrates, instead, on the overview of this research problem. Readers familiar with the general methodology may refer directly to other chapters for discussions of the various issues.

The problem chosen for this effort was both large and complex which increased the difficulty of managing its completion. This research does not provide specific answers to specific problems, but leads to understanding - which leads to knowledge. Since the knowledge in two separate fields of study were intermingled in this work, it was important to employ a systems approach to problem solving. The systems methodology consisted of factoring the research problem into a series of subproblems or components. This methodology then solved each subproblem in a step by step manner, each step building upon the previous component. This chapter describes the systematic approach used and outlines the steps pursued in this research.

Understanding Expert Systems

The first task was to understand the complexities of expert system construction. The expert system must aid a military analyst conducting a conflict analysis. Table III-1 lists the steps normally associated with expert systems

development [12:5-1]. The product of this development was a tool which more quickly and precisely performed many of the mundane tasks in the conflict analysis methodology. Many facets of this problem were studied and appropriately scaled to a reasonable level of effort. The second step in the development of a computer tool for conflict analysis was the formulation of the system requirements.

Table III-1. Expert System Development Strategy

1.	Analyze the System Requirements
2.	Acquire the Knowledge
3.	Design the Expert System
4.	Build the Knowledge Base File
5.	Evaluate the Expert System

System Requirements

The system requirements provide a guide that maps the input into outputs and identifies the processing functions. The real purpose in constructing the system requirements is to define in tabular form the data and processes that the prototype design must implement (see tables in chap V). Development of the system requirement tables enforces a structure on the development process. It precisely defines what the system must do. For example, the user (conflict analyst) must provide information about the conflict such as the players, options, and preferences. The system requirement tables define exactly the form for these inputs (see table V-2 in chapter V). At the culmination of the system processing, the user expects a stability analysis with possible equilibriums. Again, the system requirements specify

the form and processes needed to derive these outputs. Clearly defined system requirements also provide the criteria for making important design decisions. Decisions on the selection of expert tools and programming languages are constrained by the system requirements. Often the system requirements may grow or change during the development. The system requirements are provided in chapter V along with the prototype design tradeoffs.

Design Tradeoff Analysis

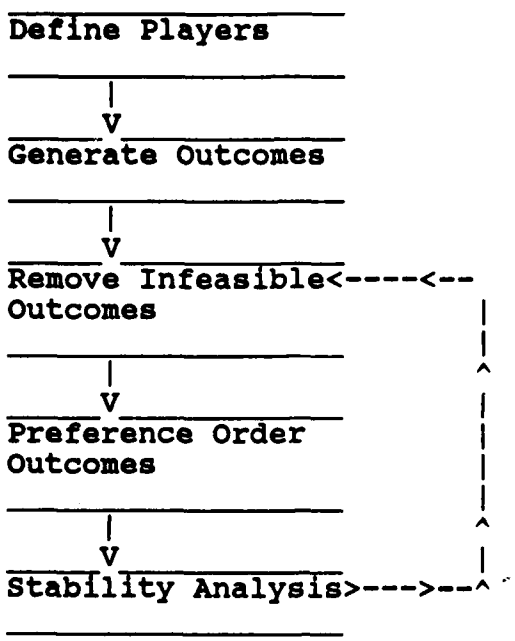
As with any software development, tradeoffs occurred frequently in the formulation of the prototype design. The goal in implementing all of the design tradeoffs was to optimize subproblems of the conflict analysis. A limiting factor, considered at this stage in the design, was the nature of a prototype system. Time and resource constraints limited the prototype in scope. The limitations while acceptable for research were not desirable for a full blown production system. Thus, the prototype system demonstrates only the feasibility of the design approach. A To extend beyond this prototype into a production system would require a different level of effort and tradeoff decisions.

The most important decision in the design phase occurred early in the research. It was the choice of the expert system development tool. Many tools were commercially available, but not necessarily available at the Air Force Institute of Technology. No one tool met all of the needs of this project and thus forced a choice on how best to support the needed capabilities. In particular, the notion of an embedded expert system narrowed the field rapidly. Very few commercial tools supported the embedded design concept and only KES was available at AFIT.

Subproblems in Conflict Analysis

A review of the Fraser and Hipel text along with papers and articles of various conflicts revealed that the process consistently performed certain actions. In conflict analysis, the repetitive actions, occurred in five distinct phases of analysis. Table III-2 below provides a flow chart of the five phases and depicts their relationship. The example in Chapter II also illustrated all five phases, as they applied to the NATO scenario, ordered sequentially as the analyst might perform them. First, the collected facts about the problem, players, and options defined the conflict. As the example from Chapter II showed, culling the information or data to provide a problem structure consumed a lot of effort. Second, the conflict yielded a number of possible outcomes. The method of generating the set of all the possible outcomes varied, but a systematic approach ensured a total set without overlooking a combination. Third, the analyst eliminated infeasible outcomes from the set of all possible outcomes. Fourth, the analyst preferentially ranked the outcomes of the various players. The ranked set of outcomes then formed the preference vector for each player. Fifth, the stability process and logic rules applied to the preference vectors (set of ordered outcomes), identified possible equilibriums. At this point a sensitivity analysis repeated the process with new or reordered preference vectors.

Table III-2. Phases in a Conflict Analysis



Each of the five distinct phases represented a logical component. Each step proceeded in turn to the next, building upon the results of the previous work. Since one of the objectives in this research was the application of expert system technology to the process, it was necessary to examine each phase. Expert system technology did not work for all of the problems, only certain classes of problems. The research, therefore, tested each step for suitability to expert system development. Establishing criteria to assess the suitability of each step was difficult. The criteria depended upon the capability of the software and hardware tools chosen. As a general guide, however, the criteria applied to the subproblem areas was whether they were most suited to solution by algorithms or by heuristics [4:8]. The areas most suited to heuristics were the ones deemed promising for an expert system to solve. Not all of the steps examined were of a heuristic nature and

amenable to expert system development. Thus, to provide a useful computer tool, a combination of conventional and expert system development was necessary.

The combination of an expert system and the conventional solution logic presented unique challenges. The standard approach in most of the expert system designs relied heavily on the problem being implemented and run under the control of an expert system design tool. Design tools vary in the amount of control for display graphics, communication with external files, or execution of external functions or programs. The high degree to which the conflict analysis solution steps interrelated with each other, suggested that the expert system and nonexpert program functions needed direct communication. The most direct communication occurred when both parts existed in the same program. An expert system integrated within the conventional program and run under its control is an embedded system.

Embedded Expert System

Building the embedded expert system occurred in three phases[12:12-1]. The first phase was the development of the control program with all the logic and functions to carry out the program except for calls to the expert system (see Table III-3). The exclusion of the calls from the main program is a practice used by software developers commonly known as stubbing. The benefit of 'stubbing' in the calls to the expert system was that it allowed the program statements to be debugged without the added complication of the expert system. Theoretically, the programmer could build an embedded system in any programming language. Many languages were considered such as Fortran, Lisp, Ada, Pascal, Basic, and C. Since many parts of the problem relied on binary vector operations and matrix math, a mathematical or scientific language appeared necessary. Hardware control and use

of assembly language modules to support the screen display were included and further narrowed the language choices. However, in practice the researcher constructed the embedded expert system design most easily using the C programming language. See reference [7] for a description of the C programming language.

Table III-3. Phase I of Embedded System

```
     Flow Chart
          |
-->  Editor     :Source Code in C (stubbed calls to Expert)
^         |
|
^
|    C Compiler      :Source  >>C.obj
^         |
|
^
|-<-  Linker         :C.obj   >>File.exe (executable)
```

Conventional Program Development

The research developed the C program in a conventionally structured method. Structured programing formulated functions, arguments, types, and variables before writing any of the program code. Many of the program functions derived their basic structure from the system requirements (see Chapter V for more detail). With each function or procedure well defined, construction of the shared parameters and data was an easier task. A flow chart of the program logic was used

to give a broad outline in the development. The flow chart guided the translation of program control logic into C program source code. Source code was the form of the program as written by an editor and stored in a normal ASCII file. The C program compiler then translated the source file into an object code file, along with a log of any errors, a memory map of symbolic names, and assembly code listings. The object file produced by the compiler combined with library files and other object modules formed an input for the linker. The linker then produced an executable file which operated from the operating system command line.

Table III-4. Integrated C Program Development Steps

```
Flow Chart of Logic
        |
Program Editor              >> Source Files
        |
Source File (ASCII)
        |
C Compiler      :Source     >>Object Files
        |
Macro Assembler      :Source     >>Object Files
        |
Linker     :C.obj + ASM.obj + Expert.obj >>File.exe
```

Conventional Debugging

The verification of the proper program logic was also conventionally debugged. The verification used sample cases or reliable examples which produced known results. The sample cases allowed the comparison of the prototype's output against known results. Other methods tested the nonlogic errors. For example, the compiler flagged errors such as syntax errors and logged them in an error listing. Logic errors required changes in the program code, while either spelling or usage corrections fixed the syntax errors. The corrected source code produced a new executable file when resubmitted to the process previously described.

Knowledge Base Development

The second phase in the embedded system development produced the knowledge base for the expert system (see table IV). The development captured, wrote, and translated the expert knowledge into a source program file. The expert system then parsed the source knowledge file. The expert system parser had many features similar to those of a language compiler. It took the source knowledge base file, an ASCII readable file, and produced a binary file which was readable only by expert system program. During program execution, the embedded expert system read and used the parsed knowledge base file at the appropriate time. The knowledge base file, like any other program, required debugging. To facilitate the process, testing and development in a stand alone mode proved the most successful. The knowledge base, in much the same manner as the C program, was tested by inputting known quantities and observing the results or output.

Table III-5. Phase II of Embedded System Development

```
Editor    :Source Knowledge Base File
    |
Expert System Parser      :source >> File.pkb
    |
File.pkb  (is read at runtime by program)
    |
Expert System Program
```

Knowledge Engineering

As discussed previously, the knowledge base development occurred in the second phase of the embedded program development. The knowledge base, however, was acquired by knowledge engineering the expert or sources of expertise. The knowledge engineering phase was crucial to properly capturing the facts, heuristics, and experience of the expert. The engineering process focused on a knowledge base that operated in a manner consistent with the process that the expert used. A discussion in chapter V presents the expert system knowledge base for this thesis in greater detail along with the steps taken to verify the knowledge.

The expert and the knowledge engineer are usually not the same individual. However, for this thesis the author served as both. Where outside expertise was needed Colonel O'Connell, the AFIT instructor for conflict analysis, provided additional guidance. The knowledge base construction also benefited from information provided in the Fraser and Hipel text.

Integration

The third and final step in the integration was the testing of the embedded knowledge base with the C program previously developed. The communication and calls to the expert systems were debugged, again with the aid of known cases or results. After all the parts were in place, the process rapidly evolved as changes were made to improve the speed, screen displays, or operation. Because the different parts of the program were so highly interdependent, it was necessary to test the changes in stand alone mode. This followed much the same process as the development of the initial program. After the changes were functioning properly, they were recombined in the integrated program and the cycle repeated.

As the development proceeded, the C language program was supplemented with assembly language modules for improved screen control and speed. The process for their development was similar in that the assembly modules were stubbed in the C program. Assembly modules for hardware detection, screen control, and speed up were produced, debugged, and compiled separately. Many supporting debugging tools were used. The many tools used, allowed a timely completion and much of the work could not have been done without them. When the assembly modules were functioning properly they were added as objective modules and linked with the C modules produced by the compiler to form the final integrated program. The resulting product, thus was a combination of C procedures and logic, assembly language extensions, and the embedded expert system program code.

Summary

The success of any research project depends in part on the development and use of sound methodology. For this research effort a systems approach was used to solve the problem. The overall problem was examined and divided into constituent parts. Some parts were selected for knowledge engineering and solution an the expert system, while others were approached with conventional program algorithm methods. Parts which were heuristic in nature were knowledge engineered to produce the knowledge base program. With the knowledge base requirements in view, the systems requirements were established and used to perform design tradeoffs. Especially important design tradeoffs such as the choice of a design tool were made early in the development. The development of the expert system and conventional C program proceeded in parallel. The parallel development facilitated the debugging. Finally the two were combined and tested with known cases to verify proper operation.

IV. Summary of Related Work

Introduction

This chapter summarizes the related work of other researchers. The focus in this section is on the specific contributions from these related works which form the theoretical underpinnings of this thesis. Since this research effort employed an interdisciplinary solution methodology, the related works come from different fields of study.

The solution methodology employed in this research was an application of artificial intelligence and an operations research methodology, conflict analysis. While elements of these subjects are discussed here, this chapter will not provide a tutorial on either subject. *The reader is assumed to be familiar with such AI topics* as production systems, forward and backward chaining, rules, frames, and constraints. For more detailed information on AI, the reader is directed to books written by Rich [9], Winston [18], Barr and Fiegenbaum [2], while Fraser and Hipel describe conflict analysis [3].

Artificial Intelligence

Artificial intelligence is a broad problem solving science that encompasses three main areas; natural language processing, robotics and pattern-recognition, and knowledged based systems often called 'expert systems'. AI has expanded rapidly in the last ten years with increased applications for military problems. This research

followed that trend and concentrated in the expert systems area for military applications.

Expert System

Rich [9:191] provides a definition for expert systems as follows:

> "An interactive computer system that performs a task normally done by a human."

From the definition, there would appear to be very little that such a system could not do. However, in practice, limitations exist upon the applicability of such systems. Only certain tasks in this research were found to be suitable for the expert system (for further discussion see Chap V). There are fundamental differences in the operation of expert systems and conventional programs. The conventional program is a product of the structured programming concept. It uses algorithms to manipulate data with tight constraints on the form of such data. The expert system on the other hand, manipulates knowledge by use of heuristics and inferential processes [17:18]. The criteria for evaluating the suitability of candidate problems to expert system development is whether the problem is most suited to algorithms or heuristic solution methods.

Expert System Development Concerns

Expert systems have unique problems which must be considered. As the name implies, an expert system must have an expert. A single expert is ideal, but multiple experts or no expert are exceptionally difficult obstacles to overcome. The expert must be able to communicate his knowledge to the knowledge engineer. The 'lack of common sense' in an expert system may foster user distrust. The finite

knowledge available to the system implies that the system is, and always will be, limited. The knowledge engineer has some latitude in constructing the systems response. This variability can mean that the system responds properly to a specific situation but not in a more general situation. The user, is therefore, uncertain as to the reliability of the system in a general case.

Expert Systems in the Military

While no AI system for conflict analysis was found, a number of AI systems are used in military decision making. One such system is KNOBS which helps mission planning at a TACC (Tactical Air Command and Control Center). The KNOBS, knowledge-based system, is a back chaining, rule based, planner developed by Mitre Corporation [17:293].

Another TACC tool, TATR (Tactical Air Targeting Recommender) was developed by the Rand Corporation. TATR assists by planning and projecting effects of airfield attacks [17:296]. After viewing TATR's projections, the original plan can be updated and improved. The system is a menu driven, rule based, forward chainer. Experienced air targeteers were knowledge engineered to produce the heuristics and rules implemented in ROSIE. ROSIE,(Rule-Oriented System for Implementing Expertise) uses an English like syntax for ease of non programmers reading and understanding the program logic.

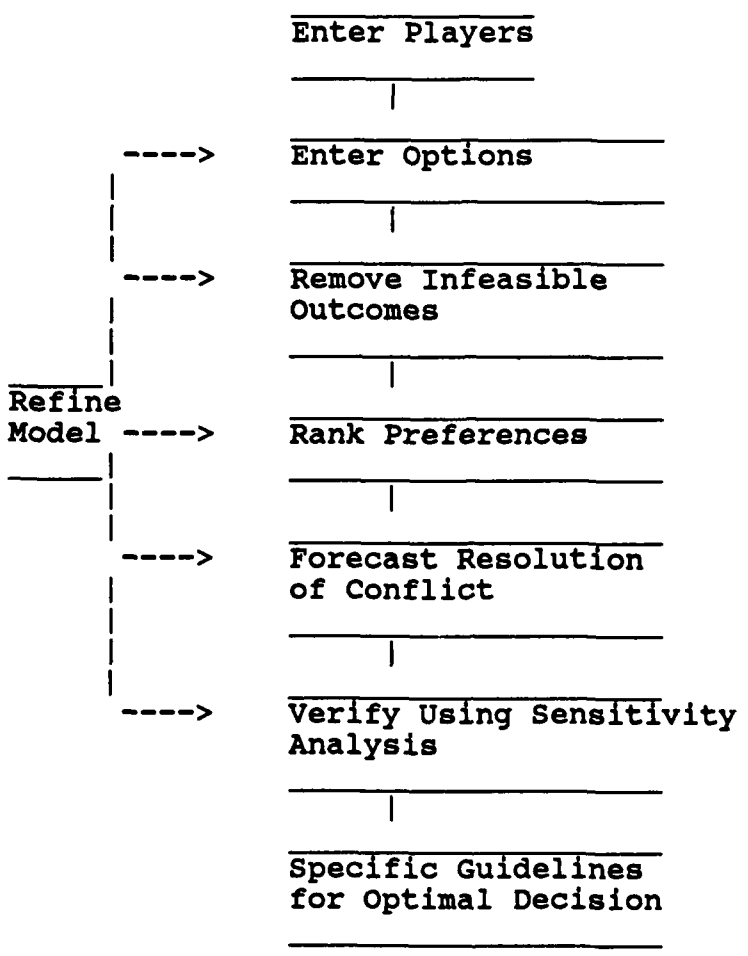
A third system provides a feature lacking in the first two, a color graphic interface. The system called SPOT was developed by SRI International. SPOT is an interactive planner for carrier aircraft launches [1:46].

Tools for Conflict Analysis

No expert systems applied to conflict analysis were found in a review of current literature. A conventional program from Waterloo Engineering Software, however, was found which performed the analysis. The program called Decisionmaker: The Conflict Analysis Program [15:1], hereafter called CAP was described as a comprehensive decision support system for strategic decision making. A demonstration version of the CAP was obtained and evaluated by the author.

The CAP written in the C programming language, runs on the IBM PC computers or systems that use the MS-DOS. The program is distributed on a 5.25 inch floppy , not as a single program but with several files. It uses the color graphics on the computer system if installed. The demonstration version of the program is restricted to a single predefined conflict and a fixed number of players. The nondemonstration version is designed for larger models with up to 10 participants, 30 options and 1,000 outcomes [15:2].

Table IV-1. CAP Program Sequence of Operations [11:3]



The CAP solution process uses 8 steps (see Table IV-1). At any stage in the process some control is afforded the user through the use of the keyboard function keys, F1-F10. The function keys summon a help menu, or allow access to other parts of the program. The first step is the entry of the players. A minimum of two up to a maximum of 10 players can be accepted. Next the user enters the options, up to a maximum of 30 total options. The options are either taken or not taken, never partially taken. Third, the infeasible outcomes are removed. The removal is done

by manually entering the options which are exclusive in a special screen display. The next stage ranks the remaining outcomes. A mask of preferred options is manually entered and the program sorts accordingly. The result of the sort is displayed and the user is allowed to move individual outcomes in the preference vector. Once the vectors are ranked, a stability anaysis is prepared and displayed. The program prompts to repeat actions for the sensitivity analysis.

The CAP program, however, has some problems. The CAP user interface is limited and it provides little user control of the screen display. While some parts of the process are prompted and occur automatically, others require more work from the user. The identification of the infeasible outcomes, for example, is entirely a user input. Once entered, the program removes the outcomes as commanded. Rank ordering of the outcomes is another area in which the user must understand the program commands. A controlling mask for the ranking operation must be entered by the user. Ideally, the program should be smarter in its handling of the ordering process and less demanding of the user.

Summary

This review examined several key AI expert system considerations. The criteria for suitability to expert system development depended upon whether heuristics or algorithms would be better. The Expert system needed a single expert who could communicate the knowledge. The expert systems have limits to their knowledge. Care must be taken to understand where the knowledge base works, and where it does not work. Three military expert system applications were discussed, although none perform conflict analysis. A conventional program for the resolution of conflict models was discussed. It performs all of the functions of a stability analysis through a traditional software design.

V. System Design

Introduction

This chapter examines the system design. The system requirements, design tradeoffs, prototype design, and evaluations will be discussed in detail. The overall process governing the flow of construction was presented in Chapter 3. The actual 'C' program code is not discussed in this chapter but is included in the appendix to this thesis.

Review of Design Process

Most successful expert systems have at least five major tasks associated with their development.
These tasks are:

1. Analyzing the Requirements
2. Acquiring the Knowledge
3. Designing the Expert System
4. Building the Knowledge Base
5. Evaluating the Expert System

The actual tasks often vary, and the sequence of task completion is subject to overlap during the development. This chapter begins with the system requirements

and draws upon the knowledge of a conflict analysis. Chapter 2 provided a detailed example of a conflict analysis and should be reference for additional information.

Analyzing the Requirements

Traditional software development and expert system design are similar in the first stage of development. General information about the problem domain must be gathered so that initial design decisions can be made. The systems tasks are established and the general expectations of the users are factored into the man/machine interface.

The system requirements in this section will cover the following areas:

1. Identify the external requirements (Task Analysis)
2. Determine the available domain resources.
3. Characterize the end users.
4. Identify the expected user environment.

While the requirements process would appear to be fixed at the conclusion of this process, it is not. Development is an adaptive process that refines and continually changes the system. One of the controlling factors is that as development proceeds, one encounters unexpected limitations in the hardware and software tools. By providing solutions, some tradeoffs are made in an adaptive manner. Thus, none of the requirements for this system are cast in concrete, instead, they continuously evolve during the adaptive design process.

External Requirements

The goal from the beginning of this research has been to aid the military analyst with conflict analysis. This research produced a prototype computer program which demonstrated the design concepts. While this program is complete, it is more limited in scope than a full production system. The term conflict analysis is more formally defined for this project as the mathematical process developed by Fraser and Hipel which extended metagame game theory (see Chapter 2 for example). Since the goal of any analysis in a conflict is to produce the winning strategy, this system aids in that quest. The conflict analysis method models the problem structure: identifies the players and their options, eliminates infeasible outcomes, rank orders outcomes, and then performs a stability analysis and identifies most likely conflict resolutions. See Table V-1 for comparison of inputs and Table V-2 for the outputs of current manual method, the prototype, and a fully developed operational system.

Table V-1. System Requirements Matrix: Inputs

REQUIREMENTS (Inputs)	MANUAL METHOD	THESIS PROTOTYPE	OPERATIONAL SYSTEM
List Players a. names b. options	Any number listed on paper	Accepts only 2 players	Accepts upto 100 and can search Database
List the infeasible outcomes	Listed on paper	Generated by Expert Sys consulting user	Generated Automatically by Expert System
Rank order preferences	Listed on paper	Sorted,ordered by computer Expert Sys consultation	Automatically sorts from Knowledge base

Inputs

The inputs for the conflict analysis actually come from extended intelligence gathering, historical study, or personal knowledge. The identification of the players may be a difficult problem in itself. For example, a hypothetical group kidnaps the head of a company and demands ransom - Who are the kidnappers? Are the kidnappers controlled by external powers? Are the kidnappers one group or several factions? Regardless of the situation, accurately characterizing the players is an intelligence function. The output of the intelligence assessment, should identify the key players. The example in Chapter 2 as a case in point, concluded that the players were the NATO alliance and the Warsaw Pact alliance in opposition. Once the players are established by the conflict analyst, the players options need to be determined. The information describing options must be conveyed from the analyst to the computer program logic. By providing structure for this information an accepted form is used. Fraser and Hipel use a binary number representation for

the various combinations of options to form the outcomes and this representation is used as the standard for this work. For example, two players with two options each can completely represent the 16 possible outcomes as follows:

Player1

a. 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

b. 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

Player2

a. 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1

b. 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

Decimal: 0 1 2 3 4 5 6 7 8 9 15

Examine the second column of numbers from top to bottom and find the number [1,0,0,0]. This outcome means that Player1 does option 'a' and Player2 does nothing. The vector [0,0,0,1] thus identifies an outcome and can be converted to a decimal number. In the case given, [0,0,0,1] becomes the number '1'. See chapter 2 for a discussion of binary and decimal numbers conversions. The user must either form the outcome on paper or input it to the computer for analysis. If a binary number is used to represent the strategies it can be shown that '2', exponentially raised to the power of the number of options (2^NUM OPT), is the total number of possible outcome combinations. However, some of the combinations may be infeasible and must be removed from the set of all possible outcomes. The user identifies the outcomes to be removed. Fraser and Hipel identify four types of infeasible combinations that can be found [3:34]. For example, [1,1,0,1] may be an impossible outcome and must be eliminated from the group of all possible outcomes. However, the user must supply the information necessary to remove

certain types of outcomes that are logically or preferentially infeasible. The last input required is the rank ordering of the vectors for each player. From the previous example we may have a decimal ranking something like:

Player1	3	7	1	0
Player2	0	3	7	3

The ranking as displayed for Player1 indicates that the binary number '3' outcome is the most preferred outcome in Player1's opinion. Player2, on the other hand, would prefer the 'do nothing' outcome [0,0,0,0] as the most desirable outcome. The user must supply the ranking, although the knowledge of the options may allow the expert system to suggest an ordering.

Table V-3. System Requirements Matrix: Process

REQUIREMENTS (Outputs)	MANUAL METHOD	THESIS PROTOTYPE	OPERATIONAL SYSTEM
List all possible outcomes	Enumerate on paper manually	Generate list by iteration	Generate by iteration
Reduced list of outcomes after infeas.	List derived on paper by searching	Search,sort 256 outcomes	Search,sort any number
List of UI's	Find by search and comparing manually	Search and compare up to 256 outcomes	Search and compare any number
List outcome as r,u,or s	Test,Compare outcomes,UIs manually	Test,compare 256 outcomes each with up to 15 UI's	Test,compare any number
Identify Equilibriums	Compare r,u,s for each outcome	Compare up to 256 r,u,s displayed	Compare any number
Help	Verbal explanation	Test inputs and respond to user request	Inferred errors from inputs, help provided

Output

The outputs from the conflict analysis are the intermediate steps from the analysis and stability results (see Table V-2 for summary). The process which map the inputs into the outputs are summarized in Table V-3. The intermediate results refer to the development of what Fraser and Hipel term unilateral improvements or "UI's". A typical decimal representation of the UI's is the decimal number under each position to which an improvement is possible, such as:

Player1	3	7	1 3	0 3 1

In this example player1 can unilaterally improve from outcome '1' to outcome '3' as the three under the 1 represents. An improvement from '0' to either '1', or '3' is also possible unilaterally. Using the UI's allows one to continue the analysis and to classify vectors for their relative stability. Again returning to the previous example, the vector for each player would be classed as rational 'r', unstable 'u', or sanctioned 's' according to the methodology of the conflict analysis. The vector would be output as follows:

Player1	r 3	r 7	u 1	u 0
Player2	r 0	r 3	u 7	r 1

An equilibrium occurs for outcomes for which both players are either an 'r' or 's'. In the example, 3 is r for both players and thus a possible equilibrium point. The stability outcome is represented as:

	E	x	x	x
Player1	r 3	r 7	u 1	u 0
Player2	r 0	r 3	u 7	r 1

Once the analysis has successfully completed all preceding steps leading up to the possible equilibriums, the user would continue to another conflict or adjust

the vectors to another form and resolve the problem. The repeated variation of the preference vectors is a common form of sensitivity analysis.

Table V-2. System Requirements Matrix: Outputs

REQUIREMENTS (Outputs)	MANUAL METHOD	THESIS PROTOTYPE	OPERATIONAL SYSTEM
List of all possible outcomes	List on paper,no limit of num. outcomes	Display list up to 256 outcomes	Displays any number of outcomes
Reduced list of outcomes after infeas.	List derived on paper no limit	Display limit 256 outcomes	Display any number
List of UI's	List on paper,no limit	Limit by screen size to 15	Display any number
List outcomes as r,u,or s	List on paper,no limit	List of up to 256 displayed	Display any number
Resolved Equilibriums	List on paper,no limit	List of up to 256 displayed	Display any number
Help	Verbal explanation	5 help screens	Context sensitive >>5 screens

End User and Environment

The prototype system is designed for the AFIT student environment. The system will be used by either students or researchers in conjunction with conflict analysis classes. The prototype system will aid in the solution of problems usually now done by hand. As a prototype, the system may be readily expanded to an

operational system of possible use by the CIA, DIA, or other intelligence analysis agencies. A limited knowledge of computer operation is essential. The typical engineering graduate student at AFIT is a sophisticated computer user and more knowledgeable than is required for this system. The user must have some understanding of the Fraser and Hipel text and should expect a friendly computer environment, highly graphic, and windowed. Context sensitive help and menu driven screens are assumed as the basic interface. Since the problem representations are highly structured and in vector form, textual displays are of little value. The user is assumed to be unfamiliar with AI or Expert System tools. Whatever operations occur, they must be highly structured and self explanatory, as might be the case in a menu driven system. Although speed is not a major consideration in this design a user becomes very bored with delays exceeding 15 seconds, thus the system must respond in less than 15 seconds or communicate at least frequently on its status.

Equipment Requirements

Two categories of equipment, computer hardware and computer software, are needed to support this research. The computer hardware consists of the electronic central processor unit (CPU) and its attendant support equipment, video monitor, input/output devices, mass storage, and power supplies. One of the major decisions in the system design was to determine what computer system would be used to develop the prototype. The choice was made to support the IBM Personal Computer or a close compatible because of the availability of computer equipment and the tools which were available to custom tailor the environment. The system could just as easily have been hosted on a VAX or mainframe system, however, the IBM PC is much more widely used in small applications. Although the target system

is the IBM PC, some of the development was conducted with the aid of a VAX 11/785, Zenith Z-248 PC., and an IBM Advanced Technology (AT) PC. Since the software needs specific hardware configurations in order to function correctly, a minimum system will require 512k bytes of random access memory (RAM), a 5.25 inch floppy drive, a fixed drive with at least 2M bytes, and a graphics monitor. The development system was connected to the VAX 11/780 via a serial modem. It required serial interfaces not needed by the target systems. The software environment was run under the Microsoft Disk Operating System (MS-DOS) version 3.0, 3.1, and 3.2. Primary system development was written in Microsoft C version 4.0 and integrated with Software A&E KES 2.3. Windows and screen control support libraries from Star Guidance Consulting were modified and used in the development.

Design Tradeoff Decisions

The decision to develop the prototype using the IBM PC computer established many limitations: the lack of virtual memory, limited storage capacity, limited processing power, and software dedicated to the Intel 8086/8088 CPU family. The IBM PC also offers many positive benefits to the development. In spite of its lack of power in comparison to larger machines, the thousands of companies that offer software in support of the IBM PC more than makes up for the loss. The support for the PCs has grown to the point where color graphic displays and easily customized user friendly interfaces are readily available.

Choice of an Expert System Tool

To build the expert system portion of the prototype, a decision was made to use a commercial expert system tool. The number of ES tools available is very large and growing larger. Many of the tools are produced to operate in very specific environments. Some tools are VAX or Symbolics computer dedicated systems while others are made for the IBM PC. Some of the companies market more than one version of their product for a variety of computers. In addition to the number of systems marketed, the author was constrained to work with tools available at AFIT. Given that it must be available at AFIT and operate on the IBM PC family of computers, the list of tools was narrowed to just four candidates (see Table V-5).

Table V-5. COMPARISON OF EXPERT SYSTEM TOOLS

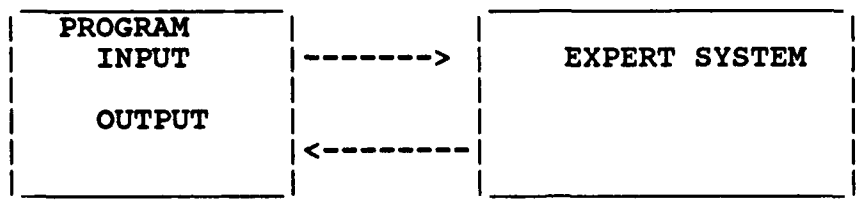
	KES (tm)	PC CONSULTANT (tm)	INSIGHT 2+ (tm)	M1 (tm)
Manufacturer:	Software A&E	Texas Instrument	Level 5 Research	Teknowledge
Price:	695	$2950	$485	$5000
Knowledge:	Rule & (class)	Rule & Frame	Rule Based	Rule & Frame
Chaining Backward:	Yes	Yes	Yes	Yes
Forward:	No	Yes	Yes	Yes
Uncertainty:	CF Bayes	CF	CF	CF
Editor:	No	Full	Yes	No
User Interface Windows:	No	Yes	Yes	Yes
Color:	No	Yes	Yes	Yes
Graphics:	No	Yes	Yes	No
Screen control:	No	Yes	Yes	Yes
Line input:	Yes	Yes	Yes	Yes
Help:	?	Yes	Yes	Yes

Table V-5. COMPARISON OF EXPERT SYSTEM TOOLS

	KES (tm)	PC CONSULTANT (tm)	INSIGHT 2+ (tm)	M1 (tm)
Externals				
Functions:	Yes	Yes	Yes	Yes
Languages:	Fortran DBMS	Scheme(tm) Dbase	Pascal Dbase	Assembler C
Embedded:	Lattice C(tm)	Lattice C(tm)	No	C or ASM
Hardware Supported				
8087:	No	?	?	?
EGA:	?	Yes	No	?
Memory:	512k	512k Extended mem used	256k	512k

The four candidate tools are: Knowledge Engineering System (KES), PC Consultant Plus (PC+), Insight 2+, and M1. All of the tools operate on the IBM PC family and are available at AFIT. The four tools considered are all ruled based systems, but only two support a frame based structure. In order to make the choice among the four tools, the system requirements were used to apply selection criteria. In particular, the weakest part in each of the tools is their lack of screen control functions. How can the outputs in Table V-2 be displayed using one of the tools? In order to evaluate the suitability of each tool, a limited review of each tools sample programs were run. None of the tools was capable of directly handling windowed screen displays of binary numbers or a stability table. Accomplishing any of the screen displays would require an external program or language to handle the input and output. PC Consultant does allow an interface to PC Scheme (a lisp programming language) that could be used. The other tools also allowed interfaces to programming languages; Assembler, Fortran, Pascal, or Scheme. The basic design concept (see Figure V-1) that emerges at this early stage is Input and Output of the analysis being performed by a program written in a language which would communicate with the expert system and return a result for display to the calling program.

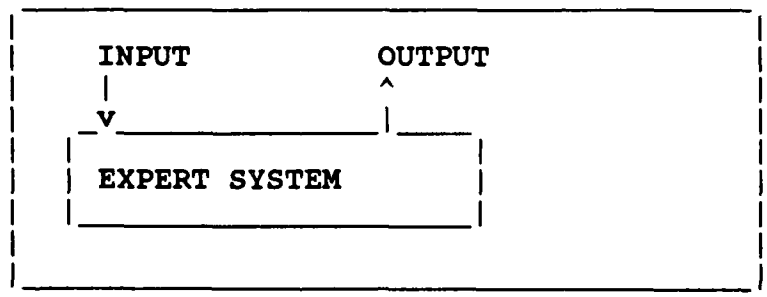
Figure V-1. External Program Communication Design



Of the inputs identified by the system requirements, not all were considered suitable for expert system development. In particular, generating, sorting, and searching binary number outcomes are better handled in the conventional programming language. Each of the inputs is a subproblem in conflict analysis as discussed in Chapter 3. They are each used in turn to construct the next step. For example, in order for the list of all possible outcomes to be generated, the list of players and options from the previous step are needed. The list of outcomes is then used in the following step when infeasible outcomes are removed. Passing of large parts of the data base or programs arrays between elements would be needed to support this operation. The external communications, thus, are a key problem in this design. One of the tools, KES, offers a unique solution to overcoming some of the external communication problems. KES provides a version of the expert tool which can be embedded in a C program. Like any subroutine or procedure, with the expert system embedded in the program, it has access to the same memory, knowledge base, and program arrays. The communication is thus internal and not passed externally in an interface through the DOS (see figure V-2). Given the advantages of using the internal method, the choice was made to use the KES tool in an embedded system application.

It should also be noted that in choosing KES, the programming language must be C. KES is distributed by Software A&E in a form suitable for use with the Lattice C compiler. AFIT, however, does not have the Lattice C compiler but uses the Microsoft C compiler instead. Software A&E generously supplied the author a special version of KES compatible with the Microsoft C.

Figure V-2. Embedded Expert System Design



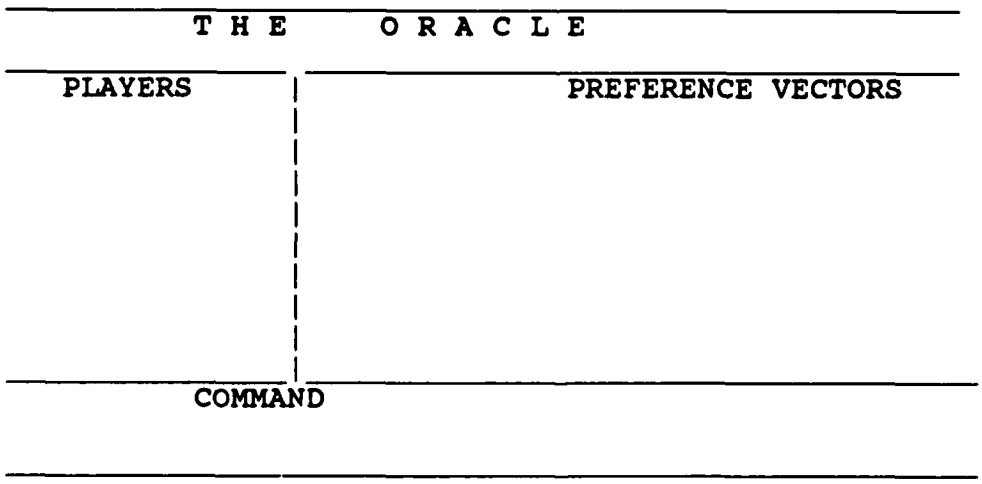
User Interface

The acceptance of a program is often based upon its user interface. Color screens, graphics, sound effects, and the general 'look and feel' of the program screen displays are important to the user. No matter how sophisticated the math or program code, the user sees only the interface. User friendliness is often quoted as desirable but difficult to quantify, measure, or provide. In designing the prototype system, the author elected to use color window displays as the primary interface. From the previous design tradeoffs the programming language was specified as C. The C language does not in itself support any input or output functions. All of the

input and output in the C language are added functions designed and implemented in libraries of support routines. The author used the Microsoft C library functions, but these do not support the color windows. In order to add a windowed capability, a support library of routines from Star Guidance Consulting (The Window Boss) were added to the standard libraries. By calling the routines in the window support library, control of the color window displays was greatly enhanced.

The main working screen is composed of four separate working windows (see Figure V-3). The title window is placed at the top, a player/option window on the left, the preference window is on the right, and at the bottom is the command window. See Chapter VI (a users guide) for more information on the use of the windows. Not displayed are the control panels and expert system communication screens. When needed, the support screens appear on the screen in response to the users commands.

Figure V-3. User Interface Display



Prototype Program Design

The main C procedures developed to implement the system requirements are provide in Table V-7. Their functions are discussed in the following paragraphs. The C source program code is provided in appendix. Each procedure in the Table V-7 corresponds to a section of the C source code.

Table V-6. Division of Duties Between C and KES

Function	C	KES	Task
Obtain Players	X		Input
Generate Outcomes	X		Process
Find infeasibles		X	Process
Remove infeas.	X		Process
Rank Preferences		X	Process
Display Vectors	X		Output
Find UIs	X		Process
Stability Analysis	X		Process
Find Equilibria	X		Process

From the inputs the user provides to the program, a system process is completed by either KES or a specific C procedure (see Table V-6). The identification of infeasible outcomes, for example, is completed by the KES section using a consultation of the user through a special window. Once the consultation has been completed, KES passes information internally back to the C program and the next procedure started. A list of the C procedures and what they do is listed in Table V-7. The internal C procedures that interface KES into the C program are listed in Table V-8. A description of each KES interface command is provided in the comments in the C source code in the appendix. For greater detail on how the commands work, the reader should refer to the KES users manual. The 'main' C

procedure is the controller for all operations and the sequence for the order in which procedures are called.

Table V-7. C Program Procedures

Procedure	Function
sound	Makes the sound effects
classic	Series of sounds to form music tune
snd_error	Series of sounds to signal error
snd_confirm	Series of sounds to signal confirmation
snd_warn	Series of sounds to signal warning
laff	Series of sounds to signal quitting
popup	Procedure creates control panel
disp_logo	Creates opening logo
disp_work	Creates the four main screens
instruct	Creates instruction display
user_help	Creates user help screen
user_colors	Creates change color screen
diagnostic	Creates diagnostic screen
con2bin	Converts decimal number to binary
con2dec	Converts binary to decimal
Generate	Creates array of outcomes
Remov_vec	Removes outcomes
Find_same	Point to outcome in other players list
stable	Find UIs
solve	Find r,s,u
simul_sanct	Check simul. sanction
equilibrium	Find equilibriums
fin_table	Create stability table
ui_review	Display all UIs
main	Controls flow of execution

Table V-8. C Procedures for Expert System Interface

Procedure	Function
KES_receive_mesg	Displays message from KES
KES_give_value_str	Gets attribute value from user
KES_g_members	Gets class members from user
KES_ld_kb	Loads knowledge base

Evaluation

In the design and construction of the prototype, three main sample inputs tested the logic. Comparison of the sample's known results with the prototypes output provided clues for debugging. In addition to debugging, correct responses from the prototype provided confidence in the system operation. The testing of specific sections of the prototype logic, required cases which used the logic in the section under test. For example, the testing of simultaneous sanctioning logic required an example that contained a simultaneous sanction in the result.

The three primary test cases originated from examples used in the Fraser and Hipel text. The first test case was the game of chicken [3:252] modeled as a conflict. Table V-9 lists the possible outcomes for the first test case. The outcomes when entered, tested the input functions of the prototype system. The stability table derived by the prototype from the inputs was compared with the known results (see Table V-10). The test case outcome '3' provided the test for the simultaneous sanction. Several iterations in the debugging process produced the proper error free logic operation. The first test case also provided checks on the functions for UI identification. Although not a very large or exhaustive test case, the first case provided a good check of communication among the various functions in the program.

Table V-9. Game of Chicken Outcomes

Player1				
	Swerve	0	1	0	1
Player2				
	Swerve	0	0	1	1
Decimal		0	1	2	3

Table V-10. Game of Chicken Stability

Player1	E	E	E	x
Swerve	r	S	r	u
	2	3	1	0
		2		1
Player2	r	S	r	u
Swerve	1	3	2	0
		1		2

The second test case reenforced the results of the first case. It exercised much of the same logic and provided additional checks on the stability. The second case, the cookie conflict [3:257], yielded a single equilibrium (see Table V-11 and Table V-12).

Table V-11. Cookie Conflict Outcomes

Child A				
	Take	0	1	0 1
Child B				
	Hit	0	0	1 1
Decimal		0	1	2 3

Table V-12. Cookie Conflict Stability Table

Child A	E	x	x	x
Take	r	u	r	u
	1	0	3	2
		1		3
Child B	r	r	u	u
	0	1	3	2
			1	0

The third test case was larger and required a far greater number of operations. Limits on the data arrays and communications between functions received further checking by case three. The third case, the Cuban Missile Crisis [3:15], also provided increased confidence in the prototype operation (see Table V-13 and Table V-14).

Table V-13. Cuban Missile Crisis Outcomes

US												
Airstrike	0	1	0	1	0	1	0	1	0	1	0	1
Blockade	0	0	1	1	0	0	1	1	0	0	1	1
USSR												
Withdraw	0	0	0	0	1	1	1	1	0	0	0	0
Escalate	0	0	0	0	0	0	0	0	1	1	1	1
Decimal	0	1	2	3	4	5	6	7	8	9	10	11

Table V-14. Cuban Missile Crisis Stability

US	E	x	x	x	x	x	x	x	x	x	x	x
Airstrike	r	s	u	u	r	u	u	u	r	u	u	u
Blockade	4	6	5	7	2	1	3	0	11	9	10	8
		4	4	4		2	2	2		11	11	11
			6	6			1	1			9	9
				5				3				10
USSR												
Withdraw	r	s	r	u	r	u	r	u	u	u	u	u
Escalate	0	4	6	2	5	1	7	3	11	9	10	8
		0		6		5		7	7	5	6	0
									3	1	2	4

The test cases all produced results in less than a second. Several runs were completed and identical results boosted the confidence in the prototype operation. Inputs to the screen updated without flicker or noticeable pause. The prototype met or exceeded the time goal of operations not lasting more than 15 seconds.

The user interface was tested by AFIT student volunteers who provided comments on the operation. The dislikes and likes were adaptively corrected and incorporated in the interface during the development. Improvements such as the

user color default control and sound effects were a direct result of the evaluation comments.

Summary

The system requirements were identified. The design of the prototype program to aid conflict analysis was outlined. A single expert system could not handle the input and output requirements. A composite design was developed in which the expert system would handle some the of the tasks and be embedded in the C program written to handle the user interface for input and output. The user interface was designed as a windowed display. The system processes were divided between the expert system and the C program. Procedures were then designed to implement the logic needed. The logic of the prototype was tested with sample cases. Evaluation the user interface was provided by selected volunteers. Likes and dislikes were adaptively corrected in the development.

VI. Users Guide to Operation

Introduction

This chapter explains how to start and use the computer program developed in this thesis. The reader is assumed to have a working knowledge of the IBM Disk Operating System (DOS) or the equivalent Microsoft version. This chapter does not provide detailed language or programming instruction but should enable a user to install and use Oracle. This program is primarily a tool for the analyst and assumes the user has a working knowledge of conflict analysis methodology.

This chapter illustrates the operation of Oracle and provides examples of DOS commands to start to it. Some confusion may result as to what is to be typed by the user, and what is to be displayed by the computer. The convention used throughout will be the user's response in italic characters, and the computers response in non-italic characters. For example, to execute the DOS command to display a directory listing of the default directory- type:

C> *DIR*

The 'C>' is the computer display of the DOS prompt character. The user types 'DIR', without the apostrophes, and presses carriage return (cr).

Installation

The conflict analysis program (Oracle) is not copy protected and may be freely copied. Since the program is not copy protected, it will run from either a hard

disk drive or floppy. However, due to its size a hard disk drive is highly recommended. Before attempting to install the Oracle, the user should ensure that the minimum essential equipment for operation are met (see Table VI-1). The Oracle is an executable file which is not in a "type" or "list" readable form. Compiled under the Microsoft C compiler, Oracle will run any any IBM PC,XT,AT or close compatible computer. It has been tested successfully on Zenith 248,158 and IBM AT,PC. Since optional installed hardware is automatically detected, differing configurations of video and communication hardware presents no problem. To best use its color windows, a color video card or EGA display is recommended, but can be operated without them. A hard drive is optional but at least 512k of memory is needed; 640k or more is highly recommended. As presently configured, memory beyond the 640k barrier is not accessed or used by this program. Oracle will operate with either IBM or Microsoft DOS but needs version 2.0 or later to reliably use its windowing displays. Operation of the program with DOS earlier than 2.0 version may produce unpredictable results and is not recommended. Oracle will automatically test the DOS version and set its video displays accordingly.

Table VI-1. Minimum Essential Equipment

Computer:	IBM PC or compatible
Video:	Color or EGA recommended
Memory:	512k minimum
Storage:	Hard Drive is recommended Floppy
DOS:	IBM DOS ver 2.0 or greater MS DOS ver 2.0 or greater

The Oracle may be copied onto the floppy or hard disk of choice with the "diskcopy" or "copy" command. For example, to copy the Oracle from a floppy in drive 'A:' to a hard disk named 'C:' and rename it to CONFLICT, type:

C> *copy A:ORACLE.EXE C:CONFLICT.EXE*

Once the Oracle is installed in the default directory it may be run by typing its file name. For example, to run the Oracle which has been copied to the hard disk drive 'C:' and named "conflict.exe", type:

C> *CONFLICT*

While the main program will operate without the knowledge base program the knowledge base file must be in the same directory as the main program file in order to use the expert knowledge. If the parsed knowledge base file is named "expert.pkb" then copy expert.pkb into the same directory as the "conflict.exe" with the same copy commands as before.

STARTUP

To begin the Oracle start the computer with normal power and "boot up" the DOS. If the terms "bootup" or DOS are not familiar to the reader, then refer to the computer operating instructions and DOS operating guide for the computer being used. After the computer is started and the DOS prompt is displayed (usually a *prompt like 'C>'), you may begin by typing:* CONFLICT *(enter-key).* If the user wishes to use some of the special features that can be set with the command line switches (see table VI-1), then in addition to typing the filename, type a space followed by the switches. For example, to start the program without any of the sound effects turned on type:

C> *CONFLICT /s*

After a brief time, the logo screen with the opening banner will appear followed by some sound effects (see Figure VI-1). When the sound effects are completed the program prompt- "Press any key to Begin" will appear at the bottom of the screen. You may press any key on the keyboard and the program will proceed to the next information screen and halt with a short beep from the speaker. The information screen contains a brief program description in a windowed display as well the system time and DOS version number (see Figure VI-2). Again "Press any key to Begin" and the program setups the work-screen for the conflict to be analyzed. If the user wishes to start up the program and go directly to the main setup screen bypassing the opening banners, then the /f command line switch (see table VI-2) can be used.

Figure VI-1. Opening Logo Screen

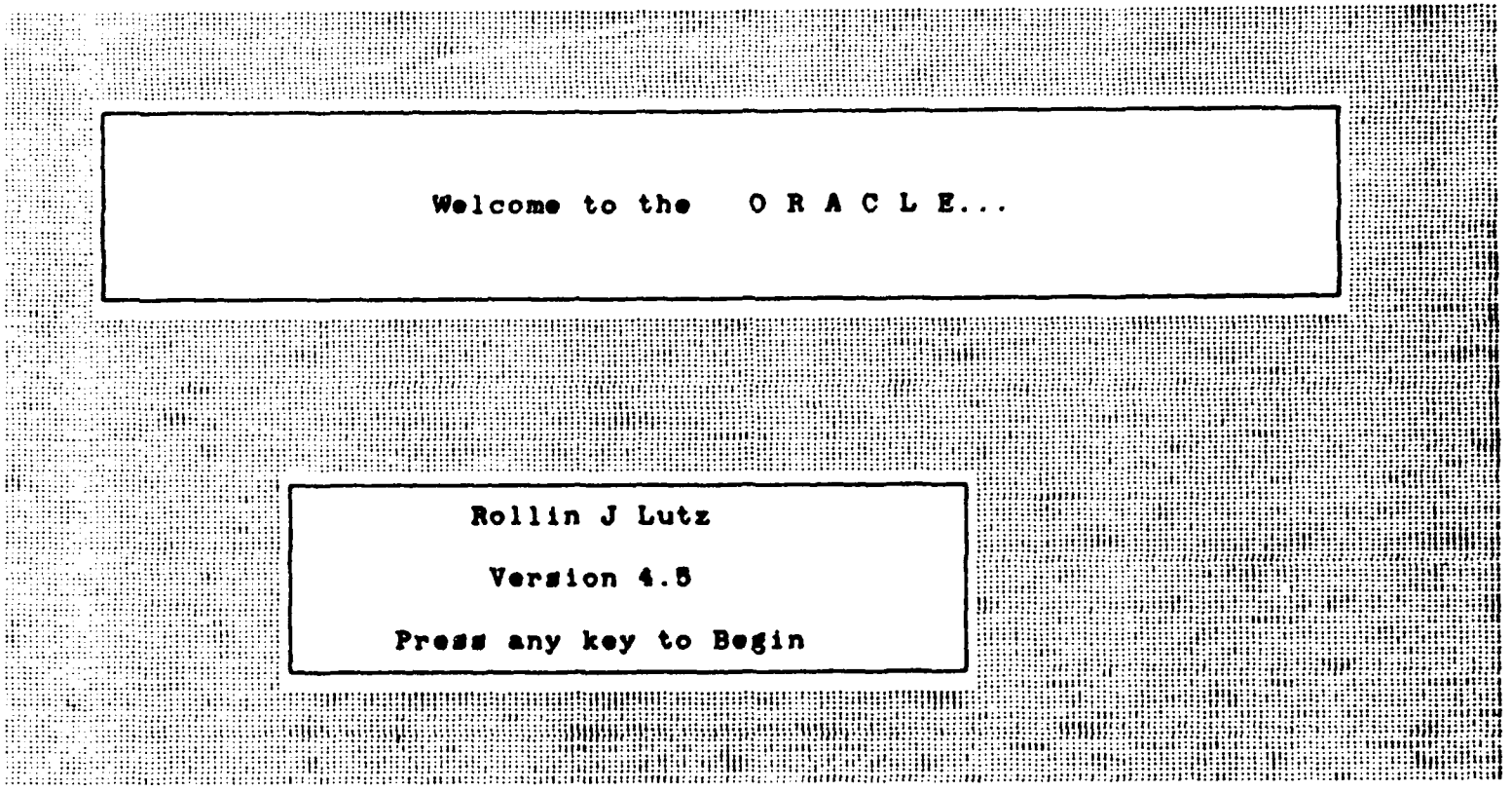
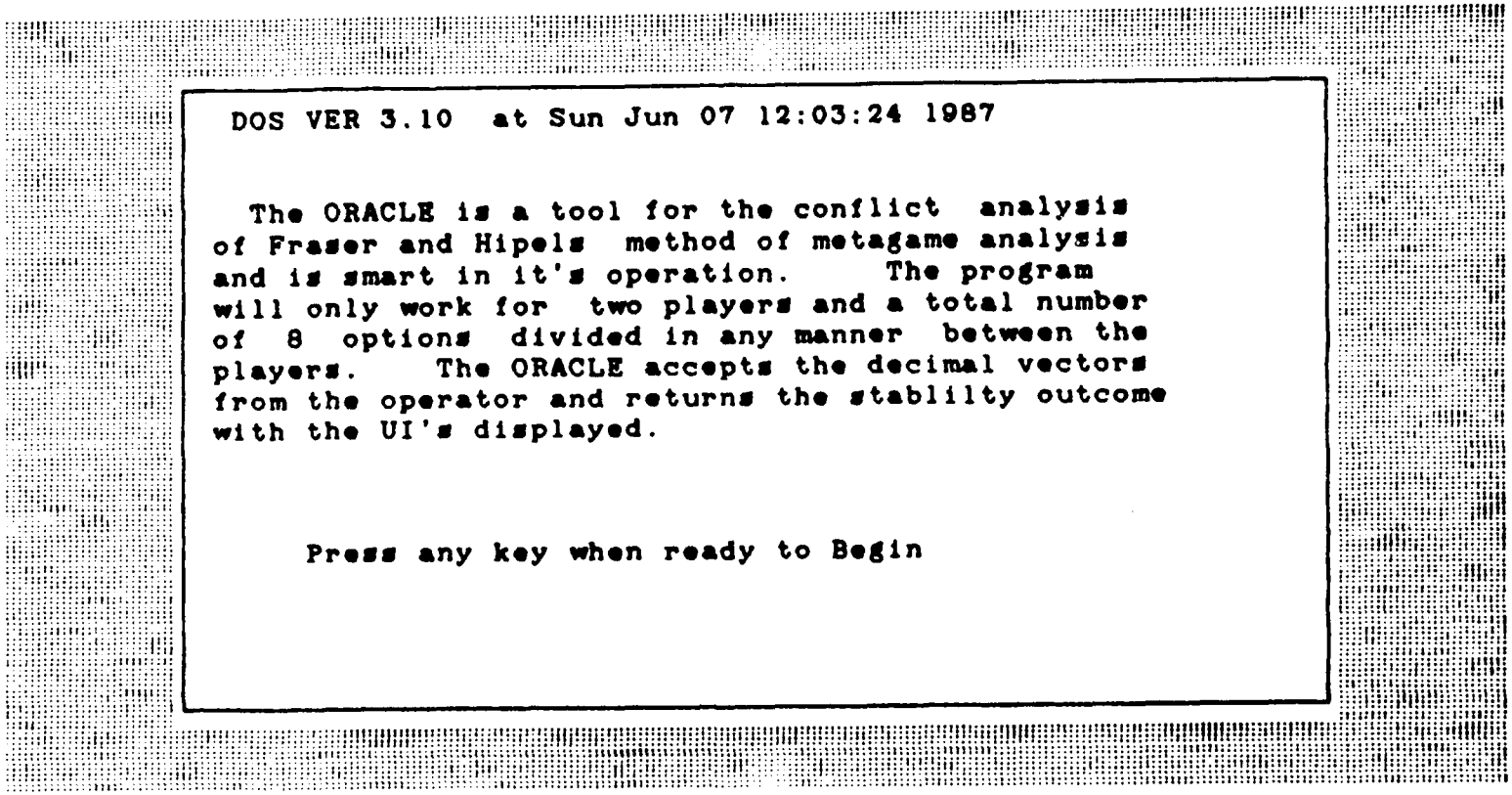


Figure VI-2. Explanation Screen



The Oracle expects to step through the analysis in an ordered manner as follows:

In a separate window on the left side of the display which is titled "Player"

1. Enter the Player1

    A. Option 1
    B. Option 2
    C. Option 3
    D. Option 4

2. Enter the Player2

    A. Option 1
    B. Option 2
    C. Option 3
    D. Option 4

3. After the players and options are entered the window directly to the right automatically displays a list of possible outcomes. Just press the (enter-key) and the window with the outcomes is cleared and the program proceeds to next step.

4. The command window at the bottom of the screen will prompt for --Enter the Preference vectors for player1. Either a decimal or binary representation is allowed.

5. After the player1 preference are entered the command window again prompts--Enter the Preference vector for Player2.

6. The program again pauses until the (enter-key) is pressed. The preference vectors will be cleared and the window will then display the Final Stability Analysis.

7. The program will pause and wait for the user to either end the session with another (enter-key) or 'h' for the help panel.

8. By using the help panel, one can revisit the various parts of the analysis. To select an option use the arrow keys on the number pad to place the cursor at the option you want to select and then press (enter-key).

9. When in doubt just follow the instruction in the command window at the bottom of the screen.

Table VI-2. Command Line Switches

SWITCH	MODE	ACTION TAKEN
/s	Silent	Disables sound effects
/f	Fast	Bypasses the opening screens
/diag	Diag	Opens a diagnostic screen
/e	Expert	Enables the expert system
/kb	Knowledge	Enables a new knowledge base file

Player1

Immediately following the instruction screen, the program sets up a title at the top and three separate windows below it. Each window serves a separate function and has its own color to distinguish it. The left side of the screen becomes the player and option window. A title at the top of the player window is displayed as "Player". The bottom five lines of the screen become a command and entry window. A title at the top is displayed as "Command" and whenever a command is prompted a short beep from the speaker follows. The right side of the screen becomes the largest window and displays the preference vectors in a form similar to the tables used by Fraser and Hipel. The program will halt automatically at places where the user is to enter or take some specific action. When the pause in execution occurs, a prompt for the desired input is displayed in the command window at the bottom of the screen along with the speaker beeping. Thus, the first pause at which a user must input some information is the command to enter the first player. The user can type from the keyboard any set of letters,(a-z, or A-Z) as legal values. Entry of a number or illegal value results in a short beep on the speaker from the program and the entry is subsequently ignored. Note, that the 'spacebar' is not a legal value, so keep the names and phrases short without spaces. The name of the first player can

be edited while being typed, with the delete and back space keys. During the entry of the name the letters appear in the player window as they are typed in the standard window colors. Once the (enter-key) is pressed, however, the name is entered, highlighted with a color change to a red background, and can not be edited with the edit keys. If an error occurs in the entry of the program players after they are entered; continue to the end of the entry stage with a series of (enter-key) and select the 'h' panel to restart the game.

After the the first player is entered and highlighted with the red background, the program halts for the first option of player 1. Again, legal values are any letter from the keyboard which are checked by the program. Since limited space for the options is available they should be kept to short phrases on the same line. When the option is entered with the (enter-key) the program steps to the next option unless it is the fourth option or an extra (enter-key) is pressed. Anytime the 'empty (enter-key)' is detected the program proceeds to the next logical step in the analysis. If in doubt about what is needed at any point in the program, check the command window for instructions.

Player2

When the command window requests the entry for player2, it is entered in the same manner as player1. The same limitations apply to the legal values. The program will only accept two players. Three or more players can not be handled by this prototype. The options may be any number from 1 to four for either player. For example, Player1 may have two options while player2 has four options. When an empty (enter-key) or the fourth option is entered the program steps to the

Preference Vector window. A short beep follows each prompt in the command window unless the silent mode (/s) was select at program startup.

Generate Outcomes

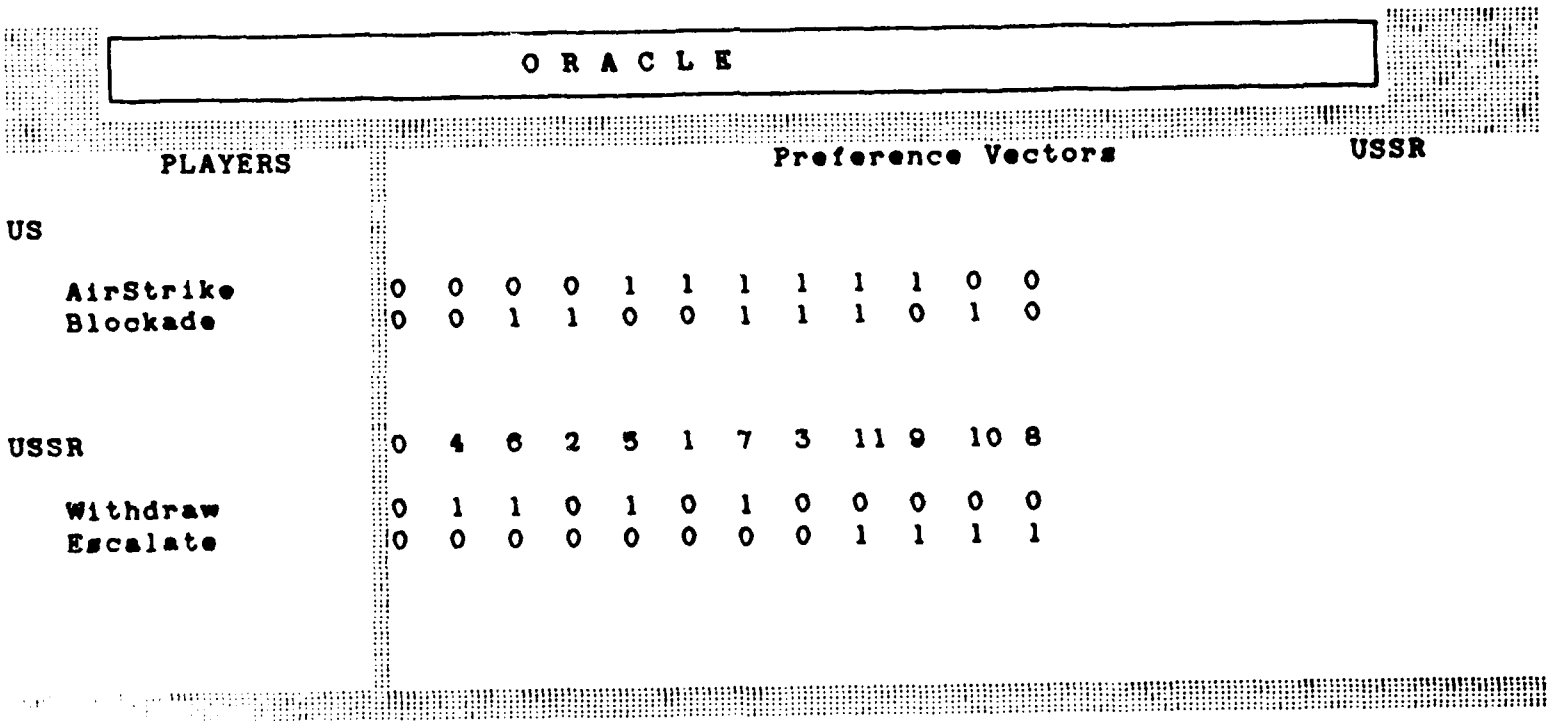
After the players and options are known to the Oracle, it generates a list of outcomes and displays them in the vector window. The outcomes are presented with the decimal value in a blue background, and the binary vector in standard colors on the screen in the row corresponding to their option. This format is the same as that used by Fraser and Hipel in the conflict analysis text. The generated outcome display is not ordered, it only shows possible outcomes.

Preference Vector Entry

The next pause in the program follows with a prompt to enter player1's preference vectors from the command window (see Figure VI-3). The player is shown in the vector window highlighted in a red background. The command window is expecting a decimal number (0-9). If a '+' is entered instead, the input switches to a binary input mode i . which only (0 or 1) is allowed. Again, legal values are checked and illegal values are beeped and ignored as before. The numbers can be edited as long as the enter-key has not been pressed, by using the back space or delete keys. After entry, the only option to correct an entry is to restart the vectors for that player with the help window, discussed below. The order in which the vectors are entered is important. The first vector is the left most vector and thus the most preferred vector. The entry proceeds in decreasing order of preference until all the vectors are in the system. An empty (enter-key) will stop the

entries and step to the next players preferences. Continue until both players are completed.

Figure VI-3. Preference Entry Screen Display

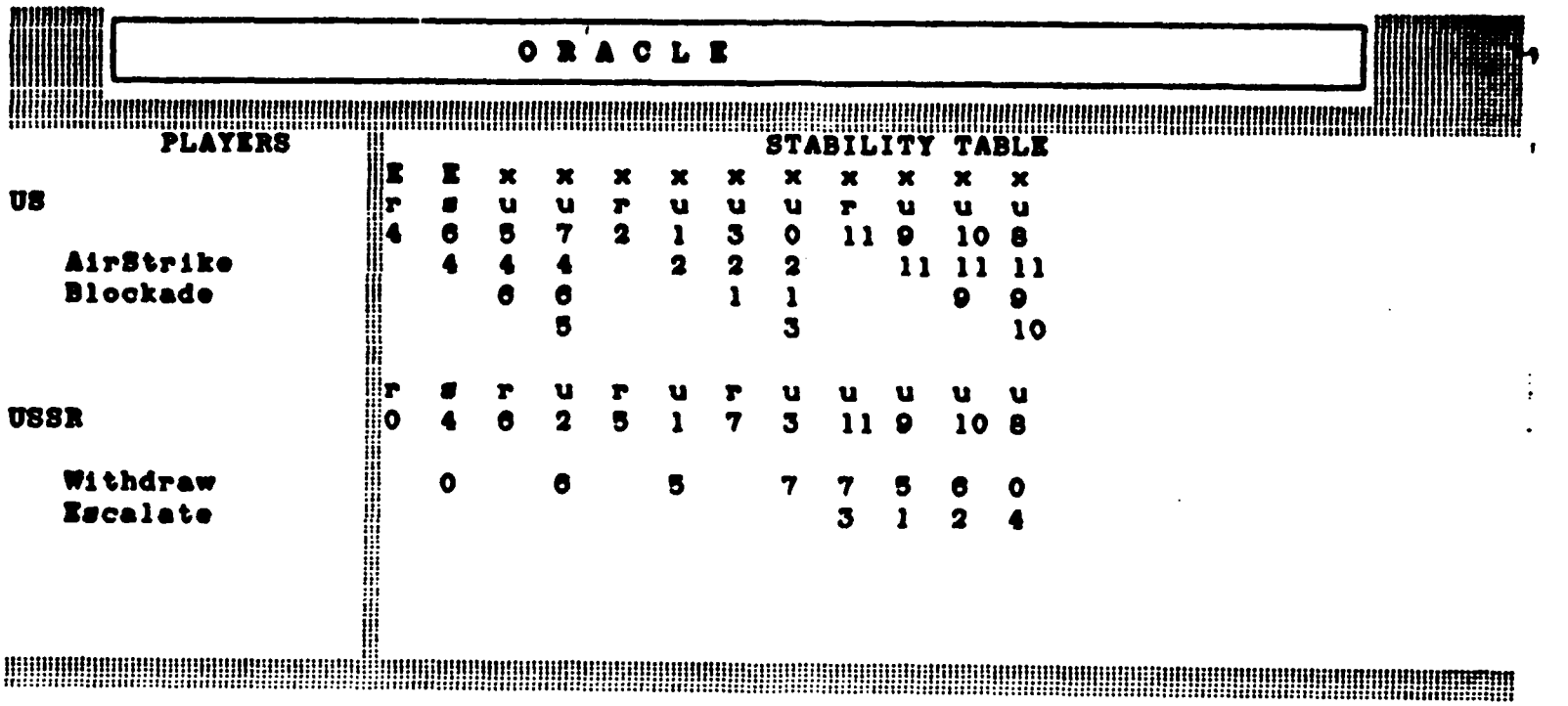


Enter VECTOR or + for binary

Stability Table

The last display in the analysis occurs after the information for the players is entered. The command window prompts for a CR,to continue or 'h' for help. The CR, to continue causes the final table to be computed and displayed in the vector window (see Figure VI-4). The equilibriums each in its own little green square, are placed above the outcomes. If it is an equilibrium then an 'E' is placed in the green square if not then 'x'. Below the equilibriums are the stability values 'r','s','u', or 'S' each of which is in its own little red background square. The r,s,u represent the same meanings as the conflict analysis conventions. The 'S' is a simultaneous sanctioned vector. Below the stability values are the decimal outcomes, and UI's. Only the first four UI's, if any, are displayed due to space available. More UI's can be seen by selecting the review UI's option from the help panel. If more than 15 outcomes are involved, only the first page of 15 is displayed. The command window will prompt for the subsequent pages to be displayed.

Figure VI-4. Stability Table Screen Display



CR--QUIT or h--HELP

Use of Expert Mode

If Oracle is started with the /e command line option the expert mode is enabled. The program will proceed to the player entry point just as before. However, once the players are entered, the outcomes are generated and a special dialog window with the expert system opens in the center of the screen (see Figure VI-5). The primary difference in the expert mode is that no manual entry of outcomes is made. The system prompts for mutually exclusive outcomes (see Figure VI-6). After a brief question and answer session, the the system removes infeasible outcomes and displays the remaining outcomes (see Table VI-7). The expert window will automatically open again. This time for ranking the outcomes into a preference vector. After the question and answer session, the preference vectors are displayed.

Once all the the outcomes are entered, the next carriage return starts the stability process. The final table is displayed. The user may then quit or use the help panel to revisit any part of the process.

Figure VI-5. Expert System Opening Screen

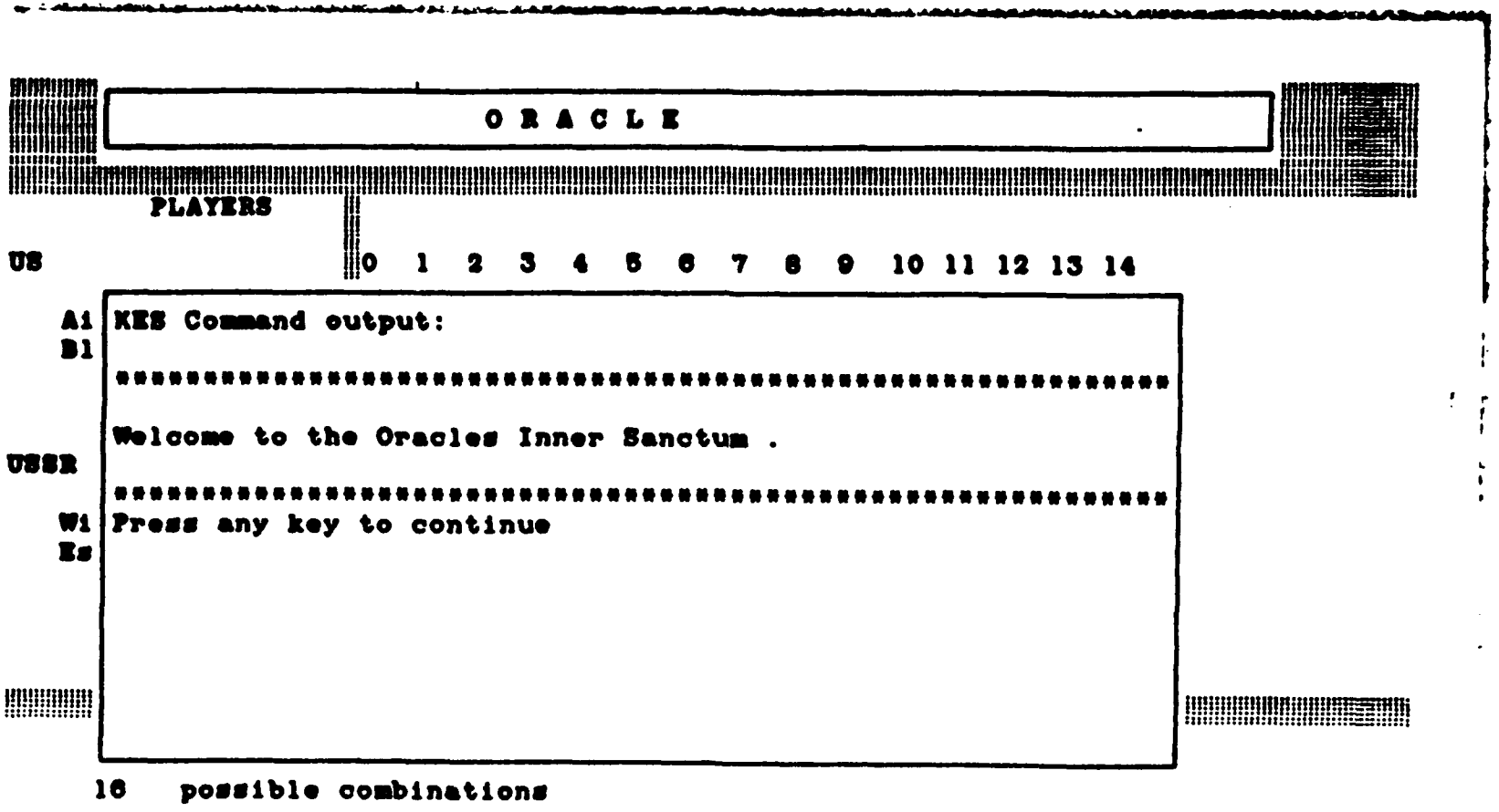


Figure VI-6. Removal of Outcomes

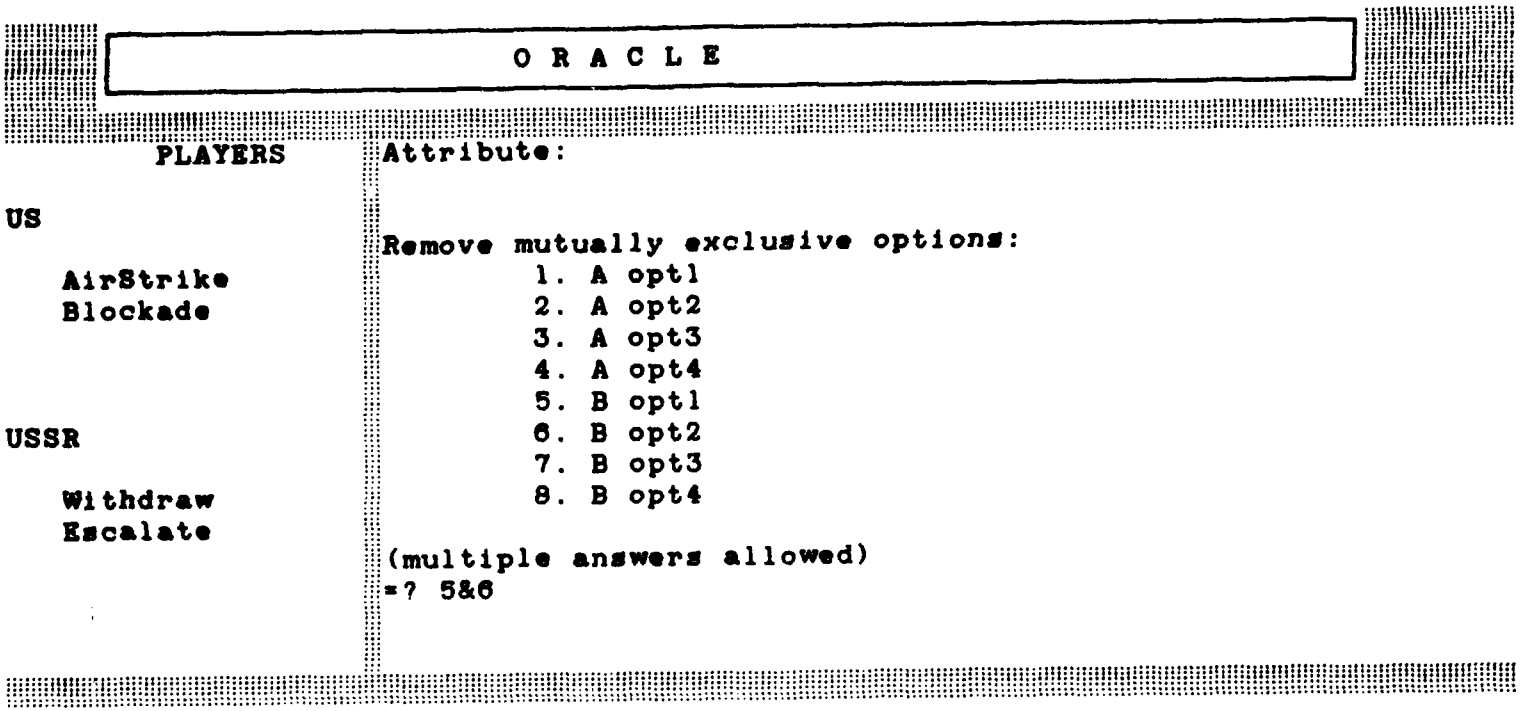
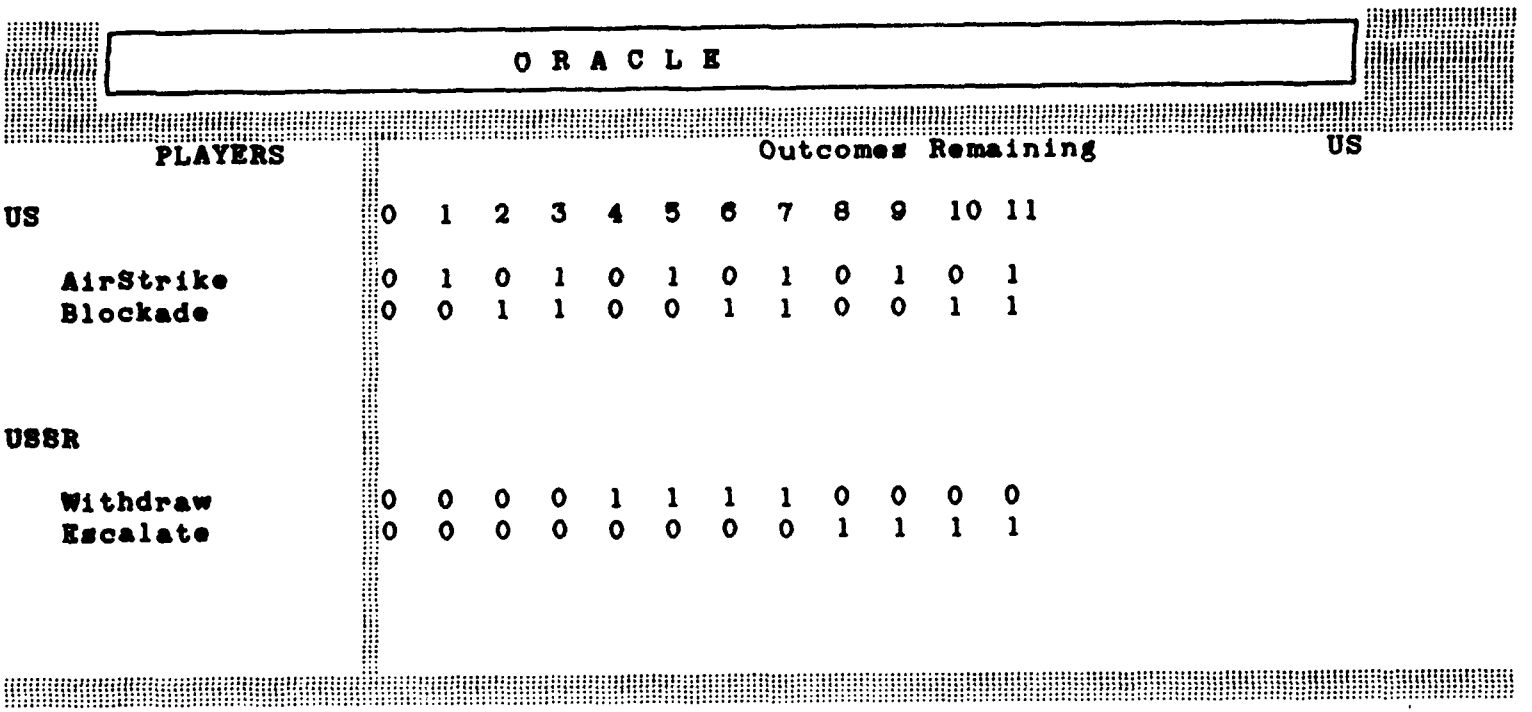


Figure VI-7. Ranking of Outcomes



CR,to rank player 1-- h for help

Help Panel

Whenever the command window at the bottom of the screen prompts "Cr,to continue -'h' for help" the help window is available. Enter the 'h', that is a lower case h with no quote marks. A popup window will open with 6 options listed (see Table VI-3). To select one of the options move the cursor with the up, down, left, right arrow keys on the key pad. When the cursor is on the desired option press the (enter-key) and the option will be executed. If no option is desired just enter the ESC key. The popup control panel will be erased, the previous screen display restored, and the speaker will beep in response to the ESC. The next command is prompted in the command window. If instead of no option, the Player1, Player2 vectors are selected the program returns to the entry stage for the appropriate player and continues forward from that point. New Game as its name implies starts a new game. Review UI's will allow a full view of UI's for both players in order then return to the command window. The help screen will display the instruction screen which has a short summary of the commands (see table VI-4). The diagnostic panel is for program configuration.

Table VI-3. Popup Control Panel

```
_________CONTROL PANEL _______________

      Player-1 Vec Reentry
      Player-2 Vec Reentry
      New Game-Start Over
      Help Screen-Commands
      Diagnostic Panel
      Review UIs-Full Page
      Expert Outcome Removal

ESC-- to quit !!
Cursor- to option, then
press RETURN for action.
```

Table VI-4. Help Screen Panel

```
_________HELP SCREEN_____________________________
DOS VER 3.30    at Thu May 21 10:12:30 1987

CONTROL COMMANDS
+     Switch to binary input
h     Get popup control panel
T     Set operation flag to True value
F     Set operation flag to False value
-     exit program and quit from vector stage
cr    Step to next program phase

COMMAND LINE SWITCHES FOR STARTUP
/e    Call KES during start
/f    Bypass the opening screens
/diag Call diagnostic screen before starting
/s    Silence the sounds
/kb filename  Use different Knowledge base

      Press any key when ready to Begin
```

Diagnostic Panel

The diagnostic panel is used in the debugging of the program but provides some user control of program features. Some of the diagnostic features can be set by command line switches before the program is started. Once the program has been started, however, the only way to dynamically change any of the features is to use the diagnostic panel (see Table VI-5). The "Call KES for expert" is one example of a feature that can be started by switch (/e) or set during program operation by setting the operation flag to a TRUE value. The flag is set by following the prompts at the bottom of the screen and entering a T for true or F for False. Once the T or F is entered the value opposite the feature is set to the appropriate value in a red highlighted box. The prompt at the bottom of the screen is erased and the next feature is prompted. The last prompt is for changing the window default color scheme.

Unlike previous features the color default prompt opens a window in the center of the screen. Three windows from the main screen, the Player, the Preference, and the Command screens can be recolored with different colors. The first screen is the player screen and is labeled as such in the new window opened. The window remains in place as the user cycles the colors by use of the arrow keys on the keyboard. The up and down arrow keys will change the foreground colors, while the left and right arrow keys change the background colors. Pressing the arrow keys will cause the window to be recolored in a new color with each press of the key. If the arrow keys are pressed enough times the colors will repeat. Pressing the up arrow for example may change to a blue which in six or seven more key presses will once again be displayed as blue. The down arrow will step back to last color displayed. The left and right arrows operate in similar manner. When the colors are finally as desired press the enter key and the color combination will be set

and the window change to the next screen to be colored. After stepping through all of the windows the program will return to the main screen in the new colors. The logo, help, and instruction windows are not user configurable.

Table VI-5. Diagnostic Panel

```
        DIAGNOSTIC PANEL
DOS VER 3.30  at Thu May 21 10:17:29 1987

Program is writing direct to memory      TRUE

Call KES for expert                      FALSE

Colors are default values                TRUE

Direct Memory Writes
Enter T,F, or cr to accept
```

Summary

This chapter has explained the features of the the Oracle. Typical user operations were explained and the various commands used in the program were given. Many of the operating screen displays were used to illustrate the options available to the user. With the explanations in this chapter a reader should be able to install, configure, and operate the Oracle.

VII. Conclusions and Recommendations

Summary of Research Effort

This research effort culminates with the development of a prototype computer program which solves conflict analysis models. The computer based tool is highly interactive and makes use of multiple windowed displays for communication. This system performs most of the tedious calculations and then displays the stability tables using color graphics for easy comprehension. This prototype uses an expert system embedded within the program that applies many facts, rules, and heuristics to the problem solving process. The embedded expert system makes the system easier to use and understand.

The prototype system can be separated into two main functional areas. The first is the embedded expert system and its contributions to the solution process. The expert system guides the analysis through consultations with the user. Once the problem is well defined in terms that the machine can use, it is fed to the conventional computational logic for resolution. The embedded expert system is able to make extensive use of the prototype's windowed displays and screen graphics. Without the embedded interface, the expert system would be a text only display. The embedded design also allows a much greater control of where and how the information is provided.

The second functional area is the conventional program logic to which the expert system feeds the data and parameters. The main program is written in the C programming language with assembly language modules to take advantage of the speed and hardware control. The bulk of the searching, sorting, and mathematical

computations are performed in this section. The conventional section is also responsible for managing the screen displays and the man/machine communication.

The successful implementation of these parts of the overall conflict analysis into the prototype answers the research problem and objective.

Intended Use of the System

The current prototype is intended to demonstrate the applicability of embedded expert system technology for aiding an operations research methodology, conflict analysis. It is not intended, as a full scale production system for use in a major problem solution. It is, however, capable of performing all the necessary steps for problems which are small enough to meet its limitations. One possible use for this prototype, is to aid students studying the conflict analysis method. Most of the problems which occur in the text for the conflict analysis [3] are small enough to be solved using the prototype. The prototype should be extended at a later date to larger problems, and rehosted on more powerful computers. Only larger computer versions will be able to handle the problems which often occur in the real world.

Comments on the C Language

Many articles have appeared recently debating the use of C as opposed to Lisp for development of expert systems. Both languages have good features, but the inability of Lisp to support the assembly language modules needed for the machine hardware is a major shortcoming. In addition, Lisp was slower in its execution while C allowed the use of machine registers for fast executions. The most attractive feature of the C code is the portability among differing CPU's. C is not without shortcomings, however. While it is a very powerful language, the C source code is

cryptic and hard to follow. This makes a C program difficult to maintain or understand. Eventually, both languages will probably support the features needed, however, for the present only C seems to have the flexibility this research needed.

Comments on KES

The focus of the development in this research was to embed the expert system within the application program. KES was the only product which supported this function. As supplied by Software A&E, KES included C language library modules that could be compiled and linked with the application. Other systems take a different approach. They prefer to either make external calls or compile and run from within the expert system. The greatest obstacle is the lack of program and screen control which this forces upon the programmer. KES has the same limitations in screen and program control when it runs as a stand alone program. However, the embedded KES is not limited except by the applications screen and program control. It functions almost like a subroutine or procedure in the program. One negative feature of KES, when embedded, is the method by which KES exits back to the calling program. When finished, the exit whether from an error condition or normal termination, forces an exit. It exits not only from KES, but the application as well. This occurs due to an undocumented feature of the 'stop' command. Removing the 'stop' from the actions section of the KES program corrects this problem.

A second undesirable feature is the disabled 'explain' and 'why' commands in the embedded mode. Normal operation of a KES consultation is greatly improved with the 'explain' command. Operation without it is possible, but not as useful.

Recommendations

This research effort should be continued through further research and development. The prototype demonstrates that the embedded expert system is a sound approach. The system is well suited to aid students in conflict analysis classes.

If KES is selected for the expert tool in future research, the developer should wait for the next version. Work performed on the older versions, may or may not be compatible with newer versions. The newer version may add improved uncertainty operations. The newer version is also reported to have corrected the problems found in this research.

Recommendations for extensions to the prototype in this research are of two types. They are C language and AI extensions. Improvements in the sorting, searching, and file operations could benefit the C execution speed. For larger problems, the computation speed becomes important. The current prototype automatically uses the math coprocessor chip if stalled. Since not all systems have the optional chip, performance can suffer. In addition, the current version of the prototype does not support printer or file operations. A feature that is needed for any serious work in future versions is printer support of the windowed displays. The user is currently limited to using DOS screen print functions for hard copy prints of the results. Unfortunately, not all computer systems support the screen print functions. Printer support is an area that would require several C language extensions. C language, or possibly assembly language modules, would be needed for save file operations. The user could store several different problems in separate files and run them without repeating the whole process.

Besides work on the C modules, the knowledge base in the AI portion can be expanded. Extensions to include conflict specific knowledge as opposed to the general problem logic should be investigated. The knowledge base can be easily

rebuilt without any changes to the prototype using the KES parser. The customized knowledge bases would increase the flexibility of the system allowing less expert analysts the benefit of the system.

Finally, one area of particular importance, would be the addition of conflict analysis hypergames. The prototype in this research did not provide any means to solve an N-level hypergame. The prototype did not allow any uncertainty in the execution of options or outcomes. This may be an artificial constraint that limits real world problems. Conflict analysis allows for problems which involve uncertainty to be modeled. One technique is the introduction of hypergames into the model. Since hypergames involve uncertainty or misinformation among players, AI would seem well suited to the problem.

Appendix A

This appendix contains the C source code for the Oracle. The C source was written using the Microsoft version 4.0 compiler. All of the main body of code is contained here. Not included in this section are any of the 'include' files with the predefined routines.

```
/*-------------------------------*/
/* Thesis Program Shell and Interface      */
/* -------------------------------*/
/*                                */
/*  Rollin J Lutz                 */
/*  28-04-87        Version 4.5   */
/*-------------------------------*/
/*  Software Tools used:          */
/*                                */
/*  C Language Compiler           */
/*    (Microsoft Ver 4.0 )        */
/*     copyright @Microsoft       */
/*                                */
/*  Window Boss                   */
/*    (Ver 3-87 )                 */
/*     copyright @Star Guidance Consulting */
/*                                */
/*                                */
/*  Knowlegde Engineering System (KES)  */
/*     (Ver 2.3 modified for MSC) */
/*     copyright @Software A&E    */
/*                                */
/*-------------------------------*/
#include "KES.H"
#include "WINDOWS.H"                /* Use screen display lib  */
#include "STDLIB.H"                 /* Use math lib functions  */
#include "TIME.H"
#include "MUSIC.H"
#define UARROW 0x48             /* define scan codes */
#define DARROW 0x50
#define LARROW 0x4b
#define RARROW 0x4d
#define ESC 0x01
#define RET   0x1c
#define SPACE  0x39
#define STRINGSIZE 50
#define MEM_SIZE 50000L

static int exp_flg = FALSE;
static int user_clrflg = FALSE;
static int kes_wn_open = FALSE;
static int fast_flag = FALSE;
static int must_close_it = FALSE;
static int silent_flg = FALSE;
static int sort_flg = FALSE;

/*-------------------------------*/
/*  Set up some of the screen colors      */
/*-------------------------------*/
static int Red_Yel = v_setatr(RED,YELLOW,0,BOLD) ;
static int Cyn_Blk = v_setatr(CYAN,BLACK,0,0) ;
static int Blu_Blu = v_setatr(BLUE,BLUE,0,BOLD) ;
static int Blu_Wht = v_setatr(BLUE,WHITE,0,0) ;
```

```
static int Blk_Wht = v_setatr(BLACK,WHITE,0,0) ;
static int Cyn_Wht = v_setatr(CYAN,WHITE,0,0) ;
static int Wht_Blk = v_setatr(WHITE,BLACK,0,BOLD) ;
static int Grn_Wht = v_setatr(GREEN,WHITE,0,BOLD);
static int Yel_Grn = v_setatr(YELLOW,GREEN,0,BOLD);
static int Red_Wht = v_setatr(RED,WHITE,0,0);
static int Grn_Blk = v_setatr(GREEN,BLACK,0,0);

/*-------------------------------*/
/*   Structure DEF for windows   */
/*-------------------------------*/
 struct mitem {
  int r;
  int c;
  char *t;
  int rv;
 };

 struct pmenu {
  int fm;
  int lm;
  struct mitem scm[25];
  };
/*-------------------------------------*/
/*  structure definition for help wind */
/*-------------------------------------*/
 static struct pmenu popinfo = {
   00, 06, {
   01, 02, "  Player-1 Vec Reentry  ", 1,
   02, 02, "  Player-2 Vec Reentry  ", 2,
   03, 02, "  New Game-Start Over   ", 3,
   04, 02, "  Help Screen-Commands  ", 4,
                    05, 02, "  Diagnostic Panel     ", 5,
   06, 02, "  Review Uls-Full Page  ", 6,
                    07, 02, "  Expert Outcome Removal", 7,
                    10, 02, "ESC --  to quit !!     ", 0,
   11, 02, "Cursor- to option, then ", 0,
   12, 02, "press RETURN for action ", 0,
   00, 00, "",99 }
  };
 WINDOWPTR w0, w1, w2, w3, w4, w5, w6, w7 ;

/*-------------------------------*/
/*   Sound routines              */
/*-------------------------------*/
sound(duration, tone)
int duration, tone;
{
        long k,l,dur;
        int l,m,n;
        outp(0x43,0xb6);
        k=0x144f38;
        l=k/tone;
        outp(0x42,l%0x100);
```

```
        outp(0x42,l/0x100);
        m=inp(0x61);
        m &= 0xff;
        n=m;
        m |= 0x3;
        outp(0x61,m);
        dur=(long)250*duration;
        for(l=0;l<=dur;l++);
        outp(0x61,n);
}
/*-------------------------------*/
/*  Produce opening Sound routines      */
/*    duration = .01 sec            */
/*    freq     = hertz              */
/*-------------------------------*/
void classic()
{
        sound(20,B) ;
        sound(20,G) ;
        sound(20,A) ;
        sound(20,B) ;
        sound(20,D) ;
        sound(20,C) ;
        sound(20,C) ;
        sound(20,E) ;
        sound(20,D) ;
        sound(20,D) ;
        sound(20,G1) ;
        sound(20,FS) ;
        sound(20,G1) ;
        sound(20,D) ;
        sound(20,B) ;
        sound(20,G) ;
        sound(20,A) ;
        sound(20,B) ;
        sound(20,C) ;
        sound(20,D) ;
        sound(20,E) ;
        sound(20,D) ;
        sound(20,C) ;
        sound(20,B) ;
        sound(20,A) ;
        sound(20,B) ;
        sound(20,G) ;
        sound(20,FSL1) ;
        sound(20,G) ;
        sound(20,A) ;
        sound(20,DL1) ;
        sound(20,FSL1) ;
        sound(20,A) ;
        sound(20,C) ;
        sound(20,B) ;
        sound(20,A) ;
        sound(20,B) ;
```

```
        sound(20,G) ;
        sound(20,A) ;
        sound(20,B) ;
        sound(20,D) ;
        sound(20,C) ;
        sound(20,C) ;
        sound(20,E) ;
        sound(20,D) ;
        sound(20,D) ;
        sound(20,G1) ;
        sound(20,FS) ;
        sound(20,G1) ;
        sound(20,D) ;
        sound(20,B) ;
        sound(20,G) ;
        sound(20,A) ;
        sound(20,B) ;
        sound(20,C) ;
        sound(20,D) ;
        sound(20,E) ;
        sound(20,D) ;
        sound(20,C) ;
        sound(20,B) ;
        sound(20,A) ;
        sound(20,B) ;
        sound(20,G) ;
        sound(20,FSL1) ;
        sound(20,G) ;
        sound(20,A) ;
        sound(20,DL1) ;
        sound(20,FSL1) ;
        sound(20,A) ;
        sound(20,C) ;
        sound(20,B) ;
        sound(20,A) ;
        sound(20,G) ;
        sound(20,DL1);
        sound(20,BL1);
        sound(20,GL1);
        sound(20,147);
        sound(20,124);
        sound(120,96);
}
/*-------------------------------*/
/*  Produce opening Sound routines      */
/*   duration = .01 sec            */
/*   freq   = hertz              */
/*-------------------------------*/
void snd_error()
{
        sound(10,440);
        sound(10,220);
}
void snd_confirm()
```

```
{
        sound(10,440);
        sound(10,880);
}
void snd_warn()
{
        sound(20,100);
}
void laff()
{
        sound(EIGHTH,C) ;
        sound(5,20000) ;

        sound(EIGHTH,C) ;
        sound(5,20000) ;

        E_REST;

        sound(QUARTER,C) ;
        sound(5,20000) ;

        sound(EIGHTH,G) ;
        sound(5,20000) ;

        sound(QUARTER,A) ;
        sound(5,20000) ;

        sound(QUARTER,FL1) ;
        sound(5,20000) ;

        Q_REST;

        sound(EIGHTH,F) ;
        sound(5,20000) ;

}
/*-------------------------------------------*/
/*      Popup function                       */
/*-------------------------------------------*/
popup(page,row,col,width,height,atrib,batrib,mx,cflag)
int page;                       /* video page */
int row, col;                   /* window - upper row/col */
int width, height;              /* window - height & width */
struct pmenu *mx;               /* pointer to popup menu struct */
int cflag;                      /* close on return strike flag */
int atrib, batrib;              /* attributes - window & border */
{
int i;                          /* scratch integer */
WINDOWPTR w;                    /* window pointer */
static WINDOWPTR wpsave;        /* a place to remember it */
int c;                          /* key scan code,,char */
static winopn = FALSE;          /* window currently open flag */
static indx = 0;                /* last open window item index */
```

```
if(winopn) {                    /* window is still open */
  goto d0;                      /* enter processing loop */
}
indx = -1;                      /* set index out of range */
w = wn_open(page, row, col, width, height, atrib, batrib);
wn_title(w,"CONTROL PANEL");
        wn_sync(w,TRUE);                /* sync cursor */
wpsave = w;                     /* save pointer */
if (w == NULL) return(99);          /* out of mem or just fubar */
winopn = TRUE;                  /* say window is open */

i = 0;                          /* init index */
while(mx->scrn[i].r) {              /* for as long as we have data */
  wn_puts(w, mx->scrn[i].r, mx->scrn[i].c, mx->scrn[i].t);
  i++;
}

d0:
 w = wpsave;                    /* restore pointer */
 i = indx;                      /* BLINDLY assume that we're back */
 if(i < mx->fm) i = mx->fm;         /* and reset if "i" is now out of */
 if(i > mx->lm) i = mx->fm;         /* range - (dumb, but it works) */
 while(TRUE) {                  /* till we exit */
  wn_locate(w, mx->scrn[i].r, mx->scrn[i].c);
  c = v_getch() >> 8;               /* see whats goin' on */
  if(c == ESC) break;               /* ESC (user wants out) - swk */
  if(c == RET) {                /* swk RETURN */
   if(cflag) {                  /* close window on return strike ? */
    wn_close(w);                /* close the window */
    winopn = FALSE;             /* say window is closed */
   }
   indx = i;                    /* remember last indx */
   return(mx->scrn[i].rv);          /* return with rv */
  }
  if(c == DARROW) c = SPACE;        /* swk conversion */
  if(c == RARROW) c = SPACE;        /* swk conversion */
  if(c == LARROW) c = BS;           /* swk conversion */
  if(c == UARROW) c = BS;           /* swk conversion */
  if(c == SPACE || c == DEL || c == BS) {
   if(c == SPACE)               /* foreward ?? */
    i++;                        /* bump index */
   else
    i--;                        /* decrement */
   if(i < mx->fm) i = mx->lm;       /* wrap on either */
   if(i > mx->lm) i = mx->fm;       /* end */
  }                             /* endif */
 }                              /* end while */
        if (silent_flg == FALSE)
        {
                snd_confirm();
}
        wn_close(w);                    /* bye bye */
winopn = FALSE;                 /* say window is closed */
return(99);                     /* nothing selected */
```

```
}
/*-------------------------------*/
/*      Open window FOR LOGO             */
/*-------------------------------*/
void disp_logo()
{
 w1 = wn_open(0,3,5,65,5,Cyn_Blk,Red_Yel);
 w2 = wn_open(0,15,15,35,5,Cyn_Blk,Red_Yel);

 wn_puts(w1,2,5,"          Welcome to the  O R A C L E...");
 wn_puts(w2,0,5,"   Rollin J Lutz  ");
 wn_puts(w2,2,10,"Version 4.5");
          if (silent_flg == FALSE)
          {
                    classic();
          }
 wn_puts(w2,4,5,"Press any key to Begin");
 v_getch();
 wn_close(w1);
 wn_close(w2);
}
/*-------------------------------*/
/* Setup Main screen area            */
/*-------------------------------*/
void disp_work()
{
          if (user_clrflg == TRUE)
          {
                    wn_clr(w2);
          wn_puts(w2,0,8,"PLAYERS");
                    wn_clr(w3);
          wn_puts(w3,0,20,"PREFERENCE VECTORS");
                    wn_clr(w4);
          wn_puts(w4,0,15,"COMMAND AREA");
          }
          else
          {
 w1 = wn_open(0,0,5,65,1,Cyn_Blk,Red_Yel);   /* title window at top */
 wn_puts(w1,0,15,"     O R A C L E");

 w2 = wn_open(1000,4,0,19,17,Blu_Wht,Blu_Blu); /* player window */
 wn_puts(w2,0,8,"PLAYERS");

 w3 = wn_open(1000,4,20,60,17,Wht_Blk,Cyn_Wht); /* vector window  */
 wn_puts(w3,0,20,"PREFERENCE VECTORS");

 w4 = wn_open(1000,22,0,80,3,Cyn_Blk,Cyn_Wht); /* command window   */
 wn_puts(w4,0,15,"COMMAND AREA");
 }
}
/*-------------------------------*/
/*  Instruction Screen               */
/*-------------------------------*/
void instruct()
```

```
{
WINDOWPTR w ;
int i, j, k, dos1, dos2 ;
long ltime;
w = wn_open(0,2,9,60,19,Wht_Blk,Blu_Wht);
time(&ltime);
wn_printf(w," DOS VER %d.%d  at %s",_osmajor,_osminor,ctime(&ltime));
wn_puts(w,3,1," The ORACLE is a tool for the conflict  analysis");
wn_puts(w,4,1,"of Fraser and Hipels  method of metagame analysis");
wn_puts(w,5,1,"and is smart in it's operation.    The program ");
wn_puts(w,6,1,"will only work for  two players and a total number");
wn_puts(w,7,1,"of  8  options  divided in any manner  between the ");
wn_puts(w,8,1,"players.   The ORACLE accepts the decimal vectors");
wn_puts(w,9,1,"from the operator and returns the stability outcome");
wn_puts(w,10,1,"with the UI's displayed. ");
        if (silent_flg = = FALSE)
        {
                and_confirm();
        }
wn_puts(w,14,1,"    Press any key when ready to Begin ");
v_getch();
wn_close(w);
}
/*-------------------------------*/
/*  Help Screen                  */
/*-------------------------------*/
void user_help()
{
WINDOWPTR w ;
int i, j, k, dos1, dos2 ;
long ltime;
w = wn_open(0,2,9,60,19,Wht_Blk,Blu_Wht);
        wn_title(w,"HELP SCREEN");
time(&ltime);
wn_printf(w," DOS VER %d.%d  at %s",_osmajor,_osminor,ctime(&ltime));
wn_puts(w,2,1,"CONTROL COMMANDS:");
wn_puts(w,3,3,"+      Switch to binary input");
wn_puts(w,4,3,"h     Get popup control panel");
wn_puts(w,5,3,"T     Set operation flag to True value");
wn_puts(w,6,3,"F     Set operation flag to False value");
wn_puts(w,7,3,"-     exit program and quit from vector stage ");
wn_puts(w,8,3,"cr    Step to next program phase ");
wn_puts(w,9,3,"&     Logical AND --> 1&2 ");
        wn_puts(w,10,3,"|     Logical OR  --> 1|2 ");
        wn_puts(w,11,3,"->    R arrow  move right  ");
        wn_puts(w,12,3,"<-    L arrow  move left   ");
        wn_puts(w,13,3,"^     Up arrow- Decimal entry in ranking ");
        wn_puts(w,14,3,"V     Down arrow - Binary entry in ranking ");
        wn_puts(w,17,1,"    Press any key for next Page ");
v_getch();
        if (silent_flg = = FALSE)
        {
                and_confirm();
        }
```

```
        wn_clr(w);
        wn_puts(w,0,1,"PAGE TWO - CONTINUED");
        wn_puts(w,1,1,"COMMAND LINE SWITCHES FOR STARTUP");
        wn_puts(w,3,1,"/e    Call KES during start");
        wn_puts(w,4,1,"/f    Bypass the opening screens");
        wn_puts(w,5,1,"/diag  Call diagnostic screen before starting");
        wn_puts(w,6,1,"/s    Silence the sounds");
        wn_puts(w,7,1,"/kb filename   Use a different knowledge base");
        wn_puts(w,8,1,"/rank  Display rank value at bottom of ranking");
        wn_puts(w,9,1,"/video Use BIOS instead of direct screen write");
 wn_puts(w,17,1,"    Press any key when ready to Begin ");
 v_getch();
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
 wn_close(w);
 }
/*-------------------------------*/
/*  Color Setup Screen           */
/*-------------------------------*/
void user_colors()
 {
 WINDOWPTR w ;
 int i, j, k, c ;
        int fore=1, back=7,Front, Back;
        static int Fcolor[8]= {BLACK,RED,GREEN,YELLOW,BLUE,MAGENTA,CYAN,WHITE};
        static int Bcolor[8]= {BLACK,RED,GREEN,YELLOW,BLUE,MAGENTA,CYAN,WHITE};
 w = wn_open(0,10,9,60,8,Red_Wht,Grn_Wht);
        wn_title(w,"SET WINDOW COLORS");
                wn_puts(w,5,1,"Press uparrow or down arrow to change Foreground");
                wn_puts(w,6,1,"Press left right arrows for Background");
                wn_puts(w,7,1,"Press return when done");

                wn_puts(w,2,1,"This sets the Player window");
        do
        {
        c = v_getch() >> 8;
        switch (c)
        {
                case RET:
                        wn_color(w2,Front,Front);
                        break;
                case UARROW:
                        ++i;
                        fore = i%8;
                        Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                        wn_natrib(w,Front);
                        break;
                case DARROW:
                        --i;
                        if (i < 0)
                                i=7;
                        fore = i%8;
```

```
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case RARROW:
                ++j;
                back = j%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case LARROW:
                --j;
                if (j < 0)
                        j=7;
                back = i%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        default:
                break;
        }
}
while (c!= RET);

        wn_clr(w);
        wn_puts(w,2,1,"This sets the Preferance Vector window");
        wn_puts(w,5,1,"Press uparrow or down arrow to change Foreground");
        wn_puts(w,6,1,"Press left right arrows for Background");
        wn_puts(w,7,1,"Press return when done");
do
{
 c = v_getch() >> 8;
 switch (c)
 {
        case RET:
                wn_color(w3,Front,Front);
                break;
        case UARROW:
                ++i;
                fore = i%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case DARROW:
                --i;
                if (i < 0)
                        i=7;
                fore = i%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case RARROW:
                ++j;
                back = j%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
```

```
                wn_natrib(w,Front);
                break;
        case LARROW:
                --j;
                if (j < 0)
                        j=7;
                back = i%8;
                Back = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        default:
                break;
        }
}
while (c!= RET);

        wn_clr(w);
        wn_puts(w,2,1,"This sets the Command window");
        wn_puts(w,5,1,"Press uparrow or down arrow to change Foreground");
        wn_puts(w,6,1,"Press left right arrows for Background");
        wn_puts(w,7,1,"Press return when done");
do
{
c = v_getch() > > 8;
switch (c)
{
        case RET:
                wn_color(w4,Front,Front);
                break;
        case UARROW:
                + +i;
                fore = i%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case DARROW:
                --i;
                if (i < 0)
                        i=7;
                fore = i%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case RARROW:
                + +j;
                back = j%8;
                Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                wn_natrib(w,Front);
                break;
        case LARROW:
                --j;
                if (j < 0)
                        j=7;
                back = i%8;
```

```
                    Front = v_setatr(Bcolor[back],Fcolor[fore],0,0);
                    wn_natrib(w,Front);
                    break;
                default:
                    break;
            }
        }
        while (c!= RET);

        wn_clr(w);
  wn_puts(w,5,1,"Press any key when ready to Begin");
  v_getch();
  wn_close(w);
 }

/*-------------------------------*/
/*  Diagnostic Screen            */
/*-------------------------------*/
void diagnostic()
 {
  WINDOWPTR w ;
  int i, j, k ;
  long ltime;
        char *user_lst;
        static char ok_lst[] = {'T','F'};
  w = wn_open(0,2,9,60,19,Wht_Blk,Blu_Wht);
        wn_title(w,"DIAGNOSTIC PANEL");
  time(&ltime);
  wn_printf(w," DOS VER %d.%d at %s",_osmajor,_osminor,ctime(&ltime));
        wn_puts(w,3,1,"Program is writing direct to memory");
        if (wn_dmaflg = = FALSE)
                wn_putsa(w,3,45,"FALSE",Red_Yel);
        else
                wn_putsa(w,3,45," TRUE",Red_Yel);

        wn_puts(w,5,1,"Call Kes for expert");
        if (exp_flg = = FALSE)
                    wn_putsa(w,5,45,"FALSE",Red_Yel);
        else
                    wn_putsa(w,5,45," TRUE",Red_Yel);
        wn_puts(w,7,1,"Colors are default values");
        wn_putsa(w,7,45," TRUE",Red_Yel);

        wn_puts(w,10,1,"Direct Memory Writes");
        wn_puts(w,11,1,"Enter T,F or cr to accept");
        wn_gets(w,user_lst,6,ok_lst);
        switch (*user_lst)
        {
                case NULL:
                    break;
                case 'T':
                    wn_dmaflg = TRUE;
                    wn_putsa(w,3,45," TRUE",Red_Yel);
                    break;
```

```
                case 'F':
                        wn_dmaflg = FALSE;
                        wn_putsa(w,3,45,"FALSE",Red_Yel);
                        break;
                default:
                        break;
                }
        wn_delrow(w,12) ;
        wn_delrow(w,11) ;
        wn_delrow(w,10) ;
        wn_puts(w,10,1,"Call Kes ");
        wn_puts(w,11,1,"Enter T,F or cr to accept");
        wn_gets(w,user_lst,6,ok_lst);
        switch (*user_lst)
        {
                case NULL:
                        break;
                case 'T':
                        exp_flg = TRUE;
                        wn_putsa(w,5,45," TRUE",Red_Yel);
                        break;
                case 'F':
                        exp_flg = FALSE;
                        wn_putsa(w,5,45,"FALSE",Red_Yel);
                        break;
                default:
                        break;
                }
        wn_delrow(w,12) ;
        wn_delrow(w,11) ;
        wn_delrow(w,10) ;
        wn_puts(w,10,1,"Change Colors ");
        wn_puts(w,11,1,"Enter T, or cr to accept defaults");
        wn_gets(w,user_lst,6,ok_lst);
        switch (*user_lst)
        {
                case NULL:
                        break;
                case 'T':
                        wn_putsa(w,7,45," USER",Red_Yel);
                        user_clrflg = TRUE;
                        user_colors();
                        break;
                default:
                        break;
                }
wn_puts(w,14,1,"   Press any key when ready to Begin ");
v_getch();
        If (silent_flg = = FALSE)
        {
                snd_confirm();
        }
wn_close(w);
}
```

```
/*-------------------*/
/* Convert decimal to binary vector */
/*-------------------*/
void con2bin(decimal,tot_num_opt,ord_num,vec)

  int vec[20][256] ;
  char *decimal ;
  int tot_num_opt, ord_num ;

  {
   int index = 1, dec_number ;
   dec_number = atoi(decimal);
   vec[0][ord_num] = dec_number ;

   do
    {
     vec[index][ord_num] = dec_number % 2 ;
     ++index ;
     dec_number /= 2 ;
    }

   while (index < tot_num_opt + 1);
  }
/*-------------------*/
/* Convert  binary vector to dec */
/*-------------------*/
void con2dec(tot_num_opt,p1_opt,horz_pos,row,vec)
        int tot_num_opt, p1_opt, horz_pos, row, vec[20][256];
        {
                int power = 1, decimal=0, space, opt, binary_bit;
                static char ok_lst[] = {'0','1','\0'};
                char *dec_vec, *ok_point;
                wn_sync(w3,TRUE);
                ok_point = ok_lst;
                for (opt=1, space=3; opt<=tot_num_opt; ++opt)
                {
                        if (opt>p1_opt)
                          space = 10 - p1_opt;
                        wn_locate(w3,opt+space,horz_pos);
                        wn_gets(w3,dec_vec,6,ok_point);
                        binary_bit = atoi(dec_vec);
                        vec[opt][row] = binary_bit;
                        decimal = decimal + (binary_bit * power);
                        power = power * 2;
                }
                vec[0][row] = decimal;
                wn_sync(w3,FALSE);
                v_hidec();
        }
/*-------------------------*/
/*   Generate all Vectors          */
/*-------------------------*/
void Generate(p1_opt,p2_opt,limit,array)
        int limit, array[20][256], p1_opt, p2_opt;
```

```
        {
        int tot_num_opt, i, j, w_limit = 15;
        int column ,index, dec_number;
        int space;
        char prn_buf[5];
        tot_num_opt = p1_opt + p2_opt;

                for (column = 0; column < limit; + + column)
                        {
                                array[0][column] = column;
                                index = 1;
                                dec_number = column;

        do
                {
                array[index][column] = dec_number % 2 ;
                + + index ;
                dec_number /= 2 ;
        }

        while (index < tot_num_opt + 1);
                        }

         for (column = 0; column < w_limit; + + column)
         {
                        sprintf(prn_buf,"%d",array[0][column]);
                  wn_putsa(w3,2,column*3,prn_buf,Blu_Wht);
                  for (i = 1, space = 3 ; i < = tot_num_opt ; + + i)
                 {
                 if (i > p1_opt)
      {
                                                space = 10 - p1_opt ;
                  }
                                               sprintf(prn_buf, "%d", array[i][column]);
                 wn_puts(w3,i + space,column*3,prn_buf);
                 }
                 }
        }
/*-------------------------------*/
/*   Outcome Removal             */
/*-------------------------------*/
int Remov_vec(array,result,limit,p1_opt)
        int array[20][256], limit, p1_opt;
        char *result;
        {
                int mask[5][2],how_many,i, j ,left;
                int pos,column, num_char, place, is_same ;
                int new_limit,where,cnt_mask;
                char con_buf[25], *temp_buf;
                if ( strncmp(result," unknown",9) = = 0)
                {
                        wn_clr(w4);
                        wn_puts(w4,1,5,"unknown removal");
                        return(limit);
```

```
        }
        sprintf(con_buf,"%s",result);
        for (left=1,pos=1,cnt_mask=0; left<9; ++left)
        {
                if (con_buf[pos] == '1')
                {
                        if (left > p1_opt)
                        {
                                where = left - p1_opt;
                        }
                        else
                        {
                                where = left;
                        }
                        mask[cnt_mask][0] = where;
                        mask[cnt_mask][1] = 1;
                        cnt_mask = cnt_mask + 1;
                }

                if (con_buf[pos] == '0')
                {
                        if (left > p1_opt)
                        {
                                where = left - p1_opt;
                        }
                        else
                        {
                                where = left;
                        }
                        mask[cnt_mask][0] = where;
                        mask[cnt_mask][1] = 0;
                        cnt_mask = cnt_mask + 1;
                }
                pos = pos + 2;
        }

        how_many = cnt_mask;
        for (column=0,place=0; column<limit; ++column)
                {
                        for (i=0,is_same=0; i<how_many; ++i)
                                {
                                        if (array[mask[i][0]][column] == mask[i][1])
                                        ++is_same;
                                }

                        if (is_same != how_many )
                                {
                                        for(j=0; j<15; ++j)
                                        {
                                                array[j][place] = array[j][column];
                                        }
                                        ++place;
                                }
}
```

```
                        return(place);
        }

/*-------------------------------*/
/*   Display Current outcomes        */
/*-------------------------------*/
void New_out(p1_opt,p2_opt,limit,array)
        int limit, array[20][256], p1_opt, p2_opt;
        {
        int tot_num_opt, i, j, w_limit;
        int column ,index, dec_number;
        int space;
        char prn_buf[5];
        tot_num_opt = p1_opt + p2_opt;

        w_limit = limit;
        if (limit > 16)
                {
                        w_limit = 16;
                }

         wn_clr(w3);
                for (column=0; column < w_limit; ++column)
         {
                        sprintf(prn_buf,"%d",array[0][column]);
                  wn_putsa(w3,2,column*3,prn_buf,Blu_Wht);
                  for (i=1, space=3 ; i<=tot_num_opt ; ++i)
                  {
                if (i > p1_opt)
      {
                                        space = 10 - p1_opt ;
                 }
                                      sprintf(prn_buf, "%d", array[i][column]);
                wn_puts(w3,i+space,column*3,prn_buf);
                 }
                        if (sort_flg == TRUE)
                        {
                                sprintf(prn_buf, "%d", array[19][column]);
                        wn_puts(w3,16,column*3,prn_buf);
                        }
                }
        }
/*-------------------------------*/
/*   Find same vector in 2          */
/*-------------------------------*/
void Find_same(Prefvec_1,Prefvec_2,same_vec,len_vec1, len_vec2)
        int Prefvec_1[20][256], Prefvec_2[20][256], same_vec[256] ;
        int len_vec1, len_vec2;
        {
                int horz_pos=0, same_pos=0 ;

                for (horz_pos=0; horz_pos < len_vec1; ++horz_pos)
                {
                        same_vec[horz_pos] = -1;  /* if no matching vec then -1 */
```

```
                    for (same_pos=0; same_pos<len_vec2; ++same_pos)
                    {
                        if (Prefvec_1[0][horz_pos] == Prefvec_2[0][same_pos])
                        {
                            same_vec[horz_pos] = same_pos;
                            break ;
                        }
                    }
                }
 }
/*-------------------------------*/
/*   Find the UI's               */
/*-------------------------------*/
void stable(start, quit, ord_num, vec)
 int start, quit, ord_num , vec[20][256] ;
 {
  int is_same , column , row , ui=10, has_ui=9 ;
                int count=0 ;

                vec[has_ui][ord_num]= 0;    /* reset number ui's to zero */
  for (column=0 ; column<ord_num ; ++column)
      {
   is_same = 0 ;
                    for (row=start ; row<=quit ; ++row )
      {
      is_same = is_same + abs(vec[row][column] - vec[row][ord_num]) ;
      }

    if (is_same == 0 )
    {
     vec[count+ui][ord_num] = vec[0][column] ;
     ++count ;
     vec[has_ui][ord_num] = count;
                                    }
   }
 }
/*-------------------------------------*/
/*   Solve Function for r,u,s          */
/*-------------------------------------*/
void solve(len_vec1,len_vec2,vec1,vec2,ans1,ans2)
 int len_vec1,len_vec2, vec1[20][256], vec2[20][256] ;
 char ans1[256], ans2[256] ;
 {
  int row, ui = 9, depth, depth2, ord_num = 0 ;
  int not_sanctioned = FALSE,place,match_vec=FALSE;
                int common_ui=FALSE;
  for (row=0 ; row<len_vec1; ++row)                    /* step through vector */
       {
   if (vec1[ui][row] == 0)
            ans1[row] = 'r';                                                    /* if no
ui then rational*/

                    else
                {
```

```
                    not_sanctioned = FALSE;      /* reset flag */
                    for (depth=ui; depth<vec1[ui][row]+ui; ++depth)          /*vec1 ui's */
            {
                /* check each ui in play A */
                                    match_vec = FALSE;
                                    for (place=0; place<len_vec2; ++place) /* check in play2 vecs */
                                    {
                                            if (vec1[depth+1][row] == vec2[0][place])                    /
* find ui in play2 */
                                                    {
                                                            match_vec = TRUE;
                                                            break;
                                                    }
                                            }
                                    if((match_vec == FALSE) || (vec2[ui][place] == 0))
                                            not_sanctioned = TRUE;
                                    else
                                    {
                                    common_ui = FALSE;                              /* reset flag for
common vectors */
                                    for (depth2=ui; depth2<vec2[ui][place]+ui; ++depth2)
                                    {
                                     match_vec = FALSE;
                                     for (ord_num=0; ord_num<len_vec1; ++ord_num)          /* find b's
ui in a*/
                                     {
                                            if (vec2[depth2+1][place] == vec1[0][ord_num])
                                                    {
                                                            match_vec = TRUE;
                                                            common_ui = TRUE;
                                                            break;
                                                    }
                                            }
        if ((match_vec == TRUE) && (row > ord_num))                                     /* is ui more
preferred */
                                    not_sanctioned = TRUE;   /* if yes then not sanctioned */
      }
                                    if (common_ui == FALSE)
                                            not_sanctioned = TRUE;  /* if no common ui then same as
no ui */
                                    }
                            }
                            if (not_sanctioned == TRUE)
                                    ans1[row] = 'u';
        else
                                    ans1[row] = 's';
                        }
                }
/*-------------------------------*/
  for (row=0 ; row<len_vec2; ++row)                 /* step through vector */
  {
                if (vec2[ui][row] == 0)
          ans2[row] = 'r';                                                          /*  if no
ui then rational*/
```

```
                else
                 {
    ord_num = 0;
                        not_sanctioned = FALSE;
                        for (depth=ui; depth<vec2[ui][row]+ui; ++depth)  /* check ui in A */
            {
                                for (place=0; place<len_vec1; ++place)
                                {
                                        if (vec2[depth+1][row] == vec1[0][place])
                                                {
                                                        match_vec = TRUE;
                                                        break;
                                                }
                                        }
                                if((match_vec == FALSE) || (vec1[ui][place] == 0))
                                        not_sanctioned = TRUE;
                                else
                                {
                                common_ui = FALSE;
                                for (depth2=ui; depth2<vec1[ui][place]+ui; ++depth2)
                                {
                                 match_vec = FALSE;
                                 for (ord_num=0; ord_num<len_vec2; ++ord_num)          /* find b's
ui in a*/
                                 {
                                        if (vec1[depth2+1][place] == vec2[0][ord_num])
                                                {
                                                        match_vec = TRUE;
                                                        common_ui = TRUE;
                                                        break;
                                                }
                                        }
          if ((match_vec == TRUE) && (row > ord_num))
                                not_sanctioned = TRUE;
       }
                                if (common_ui == FALSE)
                                        not_sanctioned = TRUE;
                                }
                        }
                        if (not_sanctioned == TRUE)
                                ans2[row] = 'u';
          else
                                ans2[row] = 's';
                        }
                }
           }
/*-------------------------------------*/
/*   Simultaneous Sanction                */
/*-------------------------------------*/
void simul_sanct(len_vec1,len_vec2,ans1,ans2,same_vec,Prefvec_1,Prefvec_2)
          int same_vec[256],Prefvec_1[20][256],Prefvec_2[20][256];
          int len_vec1, len_vec2;
          char ans1[256],ans2[256];
          {
```

```
int horz_pos,sum_vec,place,i,j,not_sanctioned=0;
int ord_num, found_it;
for (horz_pos=0; horz_pos<len_vec1; ++horz_pos) /* vec in play A */
{
 if (same_vec[horz_pos] != -1)
 {
        place = same_vec[horz_pos];
        if (ans1[horz_pos] == 'u' && ans2[place] == 'u')
        {
                not_sanctioned = FALSE;
                for (i=10; i<(10+Prefvec_1[9][horz_pos]); ++i)
                {
                        for (j=10; j<(10+Prefvec_2[9][place]); ++j)
                        {
                                sum_vec = Prefvec_1[i][horz_pos]+Prefvec_2[j][place] ;
                                sum_vec = sum_vec - Prefvec_1[0][horz_pos];
                                found_it = FALSE;
                                for (ord_num=0; ord_num<len_vec1; ++ord_num)
                                {
                                        if (sum_vec == Prefvec_1[0][ord_num])
                                        {
                                                found_it = TRUE;
                                                break;
                                        }
                                }
                                if ((found_it == TRUE) && (horz_pos> ord_num))
                                        not_sanctioned = TRUE;
                        }
                }
        if (not_sanctioned == FALSE)
                ans1[horz_pos] = 'S';

        not_sanctioned = FALSE;
        for (i=10; i<(10+Prefvec_2[9][place]); ++i)
        {
                for (j=10; j<(10+Prefvec_1[9][horz_pos]); ++j)
                {
                        sum_vec = Prefvec_1[j][horz_pos]+Prefvec_2[i][place] ;
                        sum_vec = sum_vec - Prefvec_2[0][place];
                        found_it = FALSE;
                        for (ord_num=0; ord_num<len_vec2; ++ord_num)
                        {
                                if (sum_vec == Prefvec_2[0][ord_num])
                                {
                                        found_it = TRUE;
                                        break;
                                }
                        }
                        if ((found_it == TRUE) && (horz_pos> ord_num))
                                not_sanctioned = TRUE;
                        }
                }
        if (not_sanctioned == FALSE)
                ans2[place] = 'S';
```

```
                    }
 }
 }
}
/*-------------------------------------*/
/*   Equilibrium                       */
/*-------------------------------------*/
void equilibrium(len_vec1,ans1,ans2,Ans,vec1,vec2,same_vec)
        int len_vec1 ;
        char ans1[256], ans2[256], Ans[256] ;
 int vec1[20][256], vec2[20][256], same_vec[256] ;
        {
        int step, place;
                for (step=0; step<len_vec1; ++step)
                {
                if (same_vec[step] != -1)
                {
                 place = same_vec[step];
                 if (((ans1[step] == 'r') || (ans1[step] == 's') || (ans1[step] == 'S')) &&
                        ((ans2[place] == 'r') || (ans2[place] == 's') || (ans2[place] == 'S')) )
                                Ans[step] = 'E';
                        else
                                Ans[step] = 'x';
                 }
                else
                {
                 if ((ans1[step] == 'r') || (ans1[step] == 's') || (ans1[step] == 'S'))
                        Ans[step] = 'E';
                 else
                        Ans[step] = 'x';
                }
         }
        }
/*-------------------------*/
/* Set up the final stability        */
/*-------------------------*/
void fin_table(len_vec1,len_vec2,ans1,ans2,Ans,Prefvec_1,Prefvec_2)
                int len_vec1,len_vec2;
                char ans1[256], ans2[256], Ans[256] ;
        int Prefvec_1[20][256], Prefvec_2[20][256] ;
        {
                int i, row, j, horz_pos ;
                int ui_deep, pw1;
         char pt_ans1[2], pt_ans2[2], dec_vec[5] ;



 wn_clr(w3);
 wn_puts(w3,0,20,"STABILITY TABLE");
        horz_pos = 0;

        if (len_vec1 > len_vec2)
                row = len_vec1;
        else
```

```
                row = len_vec2;

        for (j=0 ; j < row ; ++j )
{
                        horz_pos = horz_pos % 45;
                        if ((horz_pos == 0) && (j != 0) )
                                {
                                 wn_clr(w4);
                                wn_puts(w4,2,10,"Press any key for next page ");
                                wn_gets(w4,dec_vec,4,0);
                                if (*dec_vec == 'h' )
                pw1 = popup(0,5,5,33,14,Yel_Grn,Grn_Wht,&popinfo,FALSE) ;
                                        wn_clr(w3);
                                        wn_puts(w3,0,20,"STABILITY TABLE");
                                        }
                        if (j < len_vec1)
                        {
                        sprintf(pt_ans1,"%c",Ans[j]);
                        wn_putsa(w3,1,horz_pos,pt_ans1,Grn_Blk); /* Equilibrium  */

 sprintf(pt_ans1, "%c", ans1[j]);
                        wn_putsa(w3,2,horz_pos,pt_ans1,Red_Wht);

 sprintf(dec_vec, "%d", Prefvec_1[0][j]); /* decimal vector */
 wn_putsa(w3,3,horz_pos,dec_vec,Blu_Wht);

                        if (Prefvec_1[9][j] < 4)
                                ui_deep = Prefvec_1[9][j] + 10;
                        else
                                ui_deep = 14;
 for (i=10 ; i < ui_deep ; ++i )  /* put up ui's  */
 {
  sprintf(dec_vec, "%d", Prefvec_1[i][j]);
  wn_puts(w3,i-6,horz_pos,dec_vec);
 }
                }
 if (j < len_vec2)
                        {
                        sprintf(pt_ans2, "%c", ans2[j]);       /*  r,s, or u   */
                        wn_putsa(w3,8,horz_pos,pt_ans2,Red_Wht);

 sprintf(dec_vec, "%d", Prefvec_2[0][j]);  /*  decimal vector  */
 wn_putsa(w3,9,horz_pos,dec_vec,Blu_Wht);


                        if (Prefvec_2[9][j] < 4)
                                ui_deep = Prefvec_2[9][j] + 10;
                        else
                                ui_deep = 14;
        for (i=10 ; i < ui_deep ; ++i )  /* ui's     */
                {
                sprintf(dec_vec, "%d", Prefvec_2[i][j]);
                wn_puts(w3,i+1,horz_pos,dec_vec);
        }
```

```
                    }
                horz_pos = horz_pos + 3;
  }
        }
/*-------------------------------*/
/* REview the Ui Procedure       */
/*-------------------------------*/
void ui_review(len_vec1,len_vec2,ans1,ans2,Ans,Prefvec_1,Prefvec_2)
                int len_vec1,len_vec2;
                char ans1[256], ans2[256], Ans[256] ;
        int Prefvec_1[20][256], Prefvec_2[20][256] ;
        {
                int i, row, j, horz_pos ;
                int ui_deep, pw1;
         char pt_ans1[2], pt_ans2[2], dec_vec[5] ;

        wn_clr(w3);
        wn_puts(w3,0,20,"STABILITY TABLE");
                wn_putsa(w3,0,45,"Player 1",Red_Yel);
                horz_pos = 0;
                row = len_vec1;

          for (j=0 ; j < row ; ++j )
   {
                        horz_pos = horz_pos % 45;
                        if ((horz_pos = = 0) && (j != 0) )
                                {
                                 wn_clr(w4);
                                wn_puts(w4,2,10,"Press any key for next page ");
                                wn_gets(w4,dec_vec,4,0);
                                if (*dec_vec = = 'h' )
                pw1 = popup(0,5,5,33,14,Yel_Grn,Grn_Wht,&popinfo,FALSE) ;
                                                wn_clr(w3);
                                                wn_puts(w3,0,20,"STABILITY TABLE");
                                                }
                        sprintf(pt_ans1,"%c",Ans[j]);
                        wn_putsa(w3,1,horz_pos,pt_ans1,Grn_Blk);  /* Equilibrium   */

   sprintf(pt_ans1, "%c", ans1[j]);
                        wn_putsa(w3,2,horz_pos,pt_ans1,Red_Wht);

   sprintf(dec_vec, "%d", Prefvec_1[0][j]); /* decimal vector */
   wn_putsa(w3,3,horz_pos,dec_vec,Blu_Wht);

                        if (Prefvec_1[9][j] < 10)
                                ui_deep = Prefvec_1[9][j] + 10;
                        else
                                ui_deep = 20;
   for (i=10 ; i < ui_deep ; ++i )  /* put up ui's  */
   {
    sprintf(dec_vec, "%d", Prefvec_1[i][j]);
    wn_puts(w3,i-6,horz_pos,dec_vec);
   }
                        horz_pos = horz_pos + 3;
```

```
        }

         wn_clr(w4);
                wn_puts(w4,2,10,"Press any key for Player 2");
                v_getch();
        wn_clr(w3);
        wn_puts(w3,0,20,"STABILITY TABLE");
                wn_putsa(w3,0,45,"Player 2",Red_Yel);
                horz_pos = 0;
                row = len_vec2;

         for (j=0 ; j < row ; ++j )
  {
                        horz_pos = horz_pos % 45;
                        if ((horz_pos = = 0) && (j != 0) )
                                {
                                 wn_clr(w4);
                                wn_puts(w4,2,10,"Press any key for next page ");
                                wn_gets(w4,dec_vec,4,0);
                                if (*dec_vec = = 'h' )
                pw1 = popup(0,5,5,33,14,Yel_Grn,Grn_Wht,&popinfo,FALSE) ;
                                        wn_clr(w3);
                                        wn_puts(w3,0,20,"STABILITY TABLE");
                                        }
                        sprintf(pt_ans1,"%c",Ans[j]);
                        wn_putsa(w3,1,horz_pos,pt_ans1,Grn_Blk);  /* Equilibrium   */

   sprintf(pt_ans1, "%c", ans2[j]);
                        wn_putsa(w3,2,horz_pos,pt_ans1,Red_Wht);

   sprintf(dec_vec, "%d", Prefvec_2[0][j]); /* decimal vector */
   wn_putsa(w3,3,horz_pos,dec_vec,Blu_Wht);

                        if (Prefvec_2[9][j] < 10)
                                ui_deep = Prefvec_2[9][j] + 10;
                        else
                                ui_deep = 20;
   for (i=10 ; i < ui_deep ; ++i )  /* put up ui's  */
   {
    sprintf(dec_vec, "%d", Prefvec_1[i][j]);
    wn_puts(w3,i-6,horz_pos,dec_vec);
   }

                horz_pos = horz_pos + 3;
  }
        }
/*-------------------------------*/
/*  Alter the ordering of decimal vectors */
/*-------------------------------*/
void alter_vec(array,limit,player,p1_opt,p2_opt)
        int array[20][256],limit,p1_opt,p2_opt;
        char *player;
{
        int cursor_pos=0,horz_pos=0,column=0,i,j,k,num_opt;
```

```
        int page=-1,place,space,next_page,quit;
        char e,*prn_buf,*pnl_in;
        unsigned c,d;
        static char ok4_lst[]= {'0','1','2','3','4','5','6','7','8','9','+','\0'};

        num_opt = p1_opt + p2_opt;
        quit = FALSE;

        wn_clr(w3);
        wn_puts(w3,0,20,"Preference Vectors");
        sprintf(prn_buf,"%s",player);
        wn_putsa(w3,0,50,prn_buf,Red_Yel);

/*      for (column=0; column < limit; ++column) */
        while (quit == FALSE)
 {
                horz_pos = horz_pos % 45;

                if (((horz_pos == 0) && (column != 0)) || (column >= limit ))
                {
                        ++page;
                        wn_clr(w4);
                wn_puts(w4,1,5,"Use <--> ARROW keys--move to cutcome,");
                        wn_puts(w4,2,5,"Enter UPARROW-Decimal  DARROW-Binary--Cr to stop ");
                        if (silent_flg == FALSE)
                        {
                                snd_confirm();
                        }
                        wn_locate(w3,2,horz_pos);
                        wn_fixcsr(w3);
                        wn_sync(w3,TRUE);
                        next_page = FALSE;

                        while (next_page == FALSE)
                        {
                                c=v_getch() >>8;
                                wn_locate(w3,2,horz_pos);
                                switch (c )
                                {
                                        case RET:
                                        next_page = TRUE;
                                        quit = TRUE;
                                        break;

                                        case DARROW:
                                        wn_putsa(w3,2,horz_pos," ",Red_Yel);
                                        wn_locate(w3,2,horz_pos);
                                place =(page * 15) + (horz_pos / 3);
                                con2dec(num_opt,p1_opt,horz_pos, place,array);
                                        wn_puts(w3,2,horz_pos," ");
                                        wn_locate(w3,2,horz_pos);
                        sprintf(prn_buf, "%d", array[0][place]);
                                wn_putsa(w3,2,horz_pos,prn_buf,Blu_Wht);
                                        wn_locate(w3,2,horz_pos);
```

```
                    wn_fixcsr(w3);
                    wn_sync(w3,TRUE);
                    break;

                    case RARROW:
                    if (((horz_pos + 3) > = 45) && (column < = limit))
                    {
                            next_page = TRUE;
                            horz_pos = 0;
                            break;
                    }
                    if (horz_pos = = 45)
                    {
                            wn_locate(w3,2,horz_pos);
                            break;
                    }
                    horz_pos = horz_pos + 3;
                    wn_locate(w3,2,horz_pos);
                    break;

                    case LARROW:

                    if ((horz_pos = = 0) && (page > 0))
                    {
                            next_page =TRUE;
                            horz_pos=0;
                            page = (page - 2);
                            column = (page+1) * 15;
                            break;
                    }
                    if ( horz_pos = = 0)
                    {
                            wn_locate(w3,2,horz_pos);
                            break;
                    }
                    horz_pos = horz_pos - 3;
                    wn_locate(w3,2,horz_pos);
                    break;

                    case UARROW:
            place = (page * 15) + (horz_pos / 3);
                    wn_putsa(w3,2,horz_pos," ",Red_Yel);
                    wn_locate(w3,2,horz_pos);
                    wn_gets(w3,prn_buf,2,0);
                    con2bin(prn_buf,num_opt,place,array);
                    wn_puts(w3,2,horz_pos," ");
                    wn_locate(w3,2,horz_pos);
    sprintf(prn_buf, "%d", array[0][place]);
            wn_putsa(w3,2,horz_pos,prn_buf,Blu_Wht);
            for (i=1, space=3 ; i< =num_opt ; + +i)
    {
    if (i > p1_opt)
space = 10 - p1_opt ;
```

```
                                        sprintf(prn_buf, "%d", array[i][place]);
                                        wn_puts(w3,i+space,horz_pos,prn_buf);
                                                }
                                                wn_locate(w3,2,horz_pos);
                                                break;

                                                default:
                                                next_page = TRUE;
                                                quit = TRUE;
                                                break;
                                        } /* end of switch statement */
                                        }               /* end of while loop */

                                        wn_sync(w3,FALSE);
                                        v_hidec();
                                }               /* end of if statement for page */
                        if (quit == TRUE)
                        {
                                wn_clr(w3);
                                break;
                        }
                        if (horz_pos == 0)
                        {
                                wn_clr(w3);
                                wn_puts(w3,0,20,"Preference Vectors");
                                sprintf(prn_buf,"%s",player);
                                wn_putsa(w3,0,50,prn_buf,Red_Yel);
                        }
                        sprintf(prn_buf,"%d",array[0][column]);
                wn_putsa(w3,2,horz_pos,prn_buf,Blu_Wht);
                        for (i=1, space=3 ; i<=num_opt ; ++i)
  {
                        if (i > p1_opt)
                {
                                        space = 10 - p1_opt ;
   }
                                        sprintf(prn_buf, "%d", array[i][column]);
                wn_puts(w3,i+space,horz_pos,prn_buf);
                }
                        horz_pos = horz_pos + 3;
                        ++column;
 }
}
/*-------------------------------------------------------*/
/* The following routine is called when KES would generate a message that
  would appear on the terminal in standalone KES, for example, error
  messages, or display commands.  In this case, the routine simply prints the
  message and message class, and returns.  Note that no messages are expected
  to be generated by this particular application.
*/
/*-------------------------------------------------------*/
void KES_receive_mesg( text_string, text_class )
  char *text_string;                    /* The message from KES  */
  KES_msg_class_type  text_class;       /* Message classification */
```

```
{
        if (kes_wn_open = = FALSE)
  {
                w6 = wn_open(0,7,5,60,15,Red_Wht,Grn_Wht);
                kes_wn_open = TRUE;
                must_close_it=TRUE;
  }
         switch (text_class)
  {
    case KES_exec_err_c:
            wn_printf(w6,"KES System error: \n%s", text_string);
            break;
    case KES_exec_state_c:
            wn_printf(w6,"KES System status: \n%s", text_string);
            break;
    case KES_display_c:
            wn_printf(w6,"KES Command display: \n%s", text_string);
            break;
    case KES_help_c:
            wn_printf(w6,"KES Command help: \n%s", text_string);
            break;
    case KES_explain_c:
            wn_printf(w6,"KES Command explain: \n%s", text_string);
            break;
    case KES_justify_c:
            wn_printf(w6,"KES Command justify: \n%s", text_string);
            break;
    case KES_message_c:
            wn_printf(w6,"KES Command output: \n%s", text_string);
            break;
    default:
            wn_printf(w6,"Unknown KES message code: \n%s", text_string);
            break;
  }
        wn_printf(w6,"Press any key to continue \n");
        v_getch();
        wn_clr(w6);
        if (must_close_it = = TRUE)
        {
        wn_close(w6);
                kes_wn_open = FALSE;
                must_close_it=FALSE;
        }
}
/*-------------------------------------------------------*/
/*
  This routine is called by KES only when the value of an undetermined
  attribute is needed.  In this case it simply displays the same message
  KES would ask for the value of the attribute, gets a response from the
  user, and returns the input string as the value of the function. The
  returned string must be persistent across the function call, i.e., it
  must either be malloc'ed space, or be declared as a static variable.
  Note that this routine does not get called in this application, because
  all of the needed attributes are either inferred in the knowledge base,
```

```
  or are supplied values from the main program.
*/
/*-------------------------------------------------------*/

char *KES_give_value_str(attribute_descriptor)
  KES_atr_type attribute_descriptor;
{
  static char input_line[STRINGSIZE];   /* Response from the user   */
  char *prompt;                 /* Prompt for the needed atr */

          if (kes_wn_open == FALSE)
  {
                    wn_clr(w4);
                    wn_clr(w3);
                    kes_wn_open = TRUE;
                    must_close_it=TRUE;
                    wn_sync(w3,TRUE);
                    wn_wrap(w3,TRUE);
  }
  input_line[0] = '\0';               /* Start with a clean string */
  prompt = KES_g_askfor_prompt( attribute_descriptor );
  wn_printf(w3,"Attribute: \n%s", prompt );
  fflush( stdout );       /*    Keep cursor on same line as prompt */
          wn_gets(w3,input_line,1,0);
  input_line[strlen(input_line)] = '\0';  /* Remove the trailing newline */
  free (prompt);          /* Since KES_g_askfor_prompt() malloc's storage */
          if (must_close_it == TRUE)
                    {
                              wn_sync(w3,FALSE);
                              v_hidec();
                              wn_clr(w3);
                              kes_wn_open = FALSE;
                    must_close_it=FALSE;
                    }
 return (input_line);
}
/*-------------------------------------------------------*/
/*-------------------------------------------------------*/
/*
  This routine is called by KES only when the members of an undetermined
  class are needed. In this case it simply displays the same message
  KES would ask for the members of the class. Like KES_give_value_str(),
  this routine is not called in this application, since user-defined classes
  are not used in the knowledge base.
*/
/*-------------------------------------------------------*/
char *KES_g_members( class_name )
  char *class_name;
{
          static char input_line[STRINGSIZE];   /* Response from the user      */
 char     *prompt;                 /* Prompt for the needed members */
          if (kes_wn_open == FALSE)
  {
                    w6 = wn_open(0,7,5,60,15,Red_Wht,Grn_Wht);
```

```
                kes_wn_open = TRUE;
                must_close_it=TRUE;
            wn_sync(w6,TRUE);
                wn_wrap(w6,TRUE);
            }
  /* Prompt the user for the needed members */

  input_line[0] = '\0';              /* Start with a clean string */
  prompt = KES_g_prompt_class( class_name );
  wn_printf(w6,"Class Input: \n%s", prompt);
  fflush( stdout );       /*  Keep cursor on same line as prompt */

/* Get the user's response */

/*  fgets( input_line, sizeof(input_line), stdin );*/
          wn_gets(w6,input_line,1,0);
  input_line[strlen(input_line)] = '\0'; /* Remove the trailing newline */
  free (prompt);          /* Since KES_g_prompt_class() malloc's storage */

          if (must_close_it = = TRUE)
          {
                wn_sync(w6,FALSE);
                v_hidec();
                wn_clr(w6);
                wn_close(w6);
                kes_wn_open = FALSE;
          must_close_it=FALSE;
          }
          return (input_line);
}
/*-------------------------------*/
/*  Sort the outcomes into vector        */
/*-------------------------------*/
void Sort_out(array,rank,limit,p1_opt)
          int array[20][256],limit,p1_opt;
          char *rank[4];
          {
                int val,key;
                int i,j,k,p;
                int temp_array[20];
                int mask_array[4][2];
                for (i=1; i<4; ++i)
                {
                      if (strcmp(rank[i]," unknown") != 0)
                      {
                            key = atoi(rank[i]);
                            if (key < 0)
                            {
                                  mask_array[i][1] = 0;
                            }
                            else
                            {
                                  mask_array[i][1] = 1;
                            }
```

```
                                key = abs(key);
                                if (key > p1_opt)               /* adjust for less than 4 options by first player */
                                {
                                        key = key - (4 - p1_opt);
                                }
                                mask_array[i][0] = abs(key);
                        }
                }
                for (i=0; i<limit; ++i)                 /* puts relative sort value in 19 row  */
                {
                array[19][i] = 50;
                for (p=1; p<4; ++p)
                {
                        if (array[mask_array[p][0]][i] == mask_array[p][1])
                                {
                                        array[19][i] = array[19][i] - ((5 - p)*(5 - p));
                                }
                }
                }
                for (i=1; i<limit; ++i)                 /* start to move arrays into order */
                {
                        for (k=0; k<20; ++k)
                                {
                                        temp_array[k] = array[k][i];
                                }
                        j       = i-1;
                        while( (j >= 0) && (temp_array[19] < array[19][j]) )
                        {
                                for (k=0; k<20; ++k)
                                        {
                                                array[k][j+1] = array[k][j];
                                        }
                                j = j-1;
                        }
                        for (k=0; k<20; ++k)
                        {
                                array[k][j+1] = temp_array[k];
                        }
                }
        }
/*-------------------------*/
/* Load KES knowledge base and remove infeas */
/*-------------------------*/
int     exp_setup(array,limit,p1_opt)
        int array[20][256],limit, p1_opt;
        {
                int new_limit;
                char *prn_buf;
         char pnl_in[2] ;
                char *result;
                KES_atr_type Removal_vector;
                static char ok2_lst[]= {'Y','y','q','h','\0'};

                Removal_vector = KES_g_atr("removal");
```

```
                KES_run_actions();

        expert:
                KES_obtain(Removal_vector);
                result = KES_g_value_str(Removal_vector);
                new_limit= Remov_vec(array,result,limit,p1_opt);
                wn_clr(w3);
                wn_puts(w3,1,5,"Outcome Removed");
                wn_puts(w3,2,5,result);

  wn_clr(w4);
                sprintf(prn_buf,"%d",new_limit);
                wn_puts(w4,1,5,prn_buf);
                wn_puts(w4,1,15,"Remaining Outcomes in set");
                wn_puts(w4,2,5,"Remove another outcome-Y, Continue-cr");
        if (silent_flg = = FALSE)
                {
                        snd_confirm();
                }
 wn_locate(w4,2,45);
 wn_gets(w4,pnl_in,6,ok2_lst);
        switch (*pnl_in)
                {
                        case 'Y':
                                KES_erase( KES_null_atr_c);
                                free(result);           /* KES_g_value_str malloc's the space so free it */
                                limit = new_limit;
                                goto expert;
                                break;

                        case 'y':
                                KES_erase( KES_null_atr_c);
                                free(result);           /* KES_g_value_str malloc's the space so free it */
                                limit = new_limit;
                                goto expert;
                                break;

                        case '\0':
                                KES_erase( KES_null_atr_c);
                                free(result);           /* KES_g_value_str malloc's the space so free it */
                                break;

                        default:
                                KES_erase( KES_null_atr_c);
                                free(result);           /* KES_g_value_str malloc's the space so free it */
                                break;
                }
        return(new_limit);
 }
 /*-------------------------------*/
 /* Load KES knowledge base and rank outcome */
 /*-------------------------------*/
 int        rank_out(array,limit,p1_opt)
        int array[20][256],limit, p1_opt;
```

```
{
        int new_limit;
        char *rank[4];
 char pnl_in[2] ;
        KES_atr_type Rank1_vector,Rank2_vector,Rank3_vector;
        static char ok3_lst[] = {'Y','y','q','h','\0'};

        Rank1_vector = KES_g_atr("rank1");
        Rank2_vector = KES_g_atr("rank2");
        Rank3_vector = KES_g_atr("rank3");
expert1:
        KES_obtain(Rank1_vector);
        rank[1] = KES_g_value_str(Rank1_vector);
        KES_erase( KES_null_atr_c);
        KES_obtain(Rank2_vector);
        rank[2] = KES_g_value_str(Rank2_vector);
        KES_erase( KES_null_atr_c);
        KES_obtain(Rank3_vector);
        rank[3] = KES_g_value_str(Rank3_vector);
        KES_erase( KES_null_atr_c);
        Sort_out(array,rank,limit,p1_opt);
        wn_clr(w3);
        wn_puts(w3,1,5,"Rank Key for sorting");
        wn_puts(w3,2,5,rank[1]);
        wn_puts(w3,3,5,rank[2]);
        wn_puts(w3,4,5,rank[3]);
 wn_clr(w4);
        wn_puts(w4,1,5,"Rank again-Y, Continue-cr");
 if (silent_flg = = FALSE)
        {
                snd_confirm();
        }
wn_locate(w4,2,45);
wn_gets(w4,pnl_in,6,ok3_lst);
 switch (*pnl_in)
        {
                case 'Y':
                        KES_erase( KES_null_atr_c);
                        free(rank[1]);          /* KES_g_value_str malloc's the space so free it */
                        free(rank[2]);
                        free(rank[3]);
                        goto expert1;
                        break;

                case 'y':
                        KES_erase( KES_null_atr_c);
                        free(rank[1]);          /* KES_g_value_str malloc's the space so free it */
                        free(rank[2]);
                        free(rank[3]);
                        goto expert1;
                        break;

                case '\0':
                        KES_erase( KES_null_atr_c);
```

```
                                free(rank[1]);          /* KES_g_value_str malloc's the space so free it */
                                free(rank[2]);
                                free(rank[3]);
                                break;

                        default:
                                KES_erase( KES_null_atr_c);
                                free(rank[2]);
                                free(rank[3]);
                                free(rank[1]);          /* KES_g_value_str malloc's the space so free it */
                                break;
                }
        return(new_limit);
}
/*-------------------------------------*/
/*   MAIN    Program starts here       */
/*-------------------------------------*/
main(argc,argv)
 int argc;
 char *argv[];

{
 static int Prefvec_1[20][256] ;
 static int Prefvec_2[20][256] ;
        static int same_vec[256];
 static int row ;
 int limit, i, j, k, pw1;
 int choose , p1_opt, p2_opt, num_opt, space ;
        int quit, horz_pos, vert_pos, len_vec1, column, len_vec2 ;
 int *array ;
 char ans1[256], ans2[256], Ans[256] ;
 char pt_ans1[2], pt_ans2[2] ;
 char player1[20], player2[20] ;
 char option[20], testflag[8] ;
 char prn_buf[5], dec_vec[5], pnl_in[2] ;
        static char diag_flg[] = {"/diag"};
        static char ok1_lst[] = {'h','\0'};
 char *KB_FILE = {"conflict.pkb"};

/*-------------------------------------*/
/*   Test the BIOS for screen update method     */
/*-------------------------------------*/
 if (_osmajor < 3)
  wn_dmaflg = FALSE ;
/*-------------------------------------*/
/*   Test the Command Line for switches        */
/*-------------------------------------*/
        if (argc != 1)
        {
                for (i = 1; i <= argc; ++i)
                {
                        if (strcmp(argv[i],diag_flg) == 0)
                        {
                                diagnostic();
```

```
                    }
                    if (strcmp(argv[i],"/f") == 0)
                    {
                            fast_flag=TRUE;
                    }
                    if (strcmp(argv[i],"/s") == 0)
                    {
                            silent_flg=TRUE;
                    }
                    if (strcmp(argv[i],"/rank") == 0)
                    {
                            sort_flg=TRUE;
                    }
                    if (strcmp(argv[i],"/e") == 0)
                    {
                            exp_flg=TRUE;
                    }
                    if (strcmp(argv[i],"/video") == 0)
                    {
                            wn_dmaflg=FALSE;
                    }
                    if (strcmp(argv[i],"/kb") == 0)
                    {
                            exp_flg=TRUE;
                            ++i;
                            sprintf(KB_FILE,"%s",argv[i]);
                    }
            }
        }
/*-------------------------------------*/
/*  Save entry screen so we can return it when done  */
/*-------------------------------------*/
 w0 = wn_save(0,0,0,78,23);
 if (!w0)  exit(0);
/*-------------------------------------*/
/*  Checkered background pattern              */
/*-------------------------------------*/
 for(i=0; i<25; i++)
  {
   v_locate(0,i,0);
   v_wca(0, 0xb0, Wht_Blk, 80);
  }

 v_hidec();          /* turn off the cursor  */
/*-------------------------------------*/
/*       Open window FOR LOGO              */
/*-------------------------------------*/
          if (fast_flag == FALSE)
          disp_logo();
/*-------------------------------------*/
/*  Get the instructions              */
/*-------------------------------------*/
          if (fast_flag == FALSE)
          instruct();
```

```
/*-------------------------*/
/* Setup the main work screen          */
/*-------------------------*/
new_game:
 disp_work();
/*-----------------*/
/*  Set up Players      */
/*-----------------*/
 wn_puts(w4,2,5,"What is the name of the first player ?");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
        wn_sync(w2,TRUE);
 wn_locate(w2,2,0);
 wn_gets(w2, player1, 4, 0);
 wn_putsa(w2, 2, 0, player1, Red_Yel);

 for(p1_opt=0 ; p1_opt<5 ; ++p1_opt)
 {
  wn_clr(w4);
  wn_puts(w4,2,5,"Enter option-- short phrase, please");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
  wn_locate(w2, p1_opt + 4 , 3);
  wn_gets(w2,option,4,0);
  if (*option == NULL)
   break ;
  wn_putsa(w2, p1_opt+4, 3, option, Blu_Wht);
  }

 wn_clr(w4);
 wn_puts(w4,2,5,"What is the name of the second player ?");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
 wn_locate(w2,9,0);
 wn_gets(w2,player2,4,0);
 wn_putsa(w2,9,0,player2,Red_Yel);

 for(p2_opt=0 ; p2_opt < 4 ; ++p2_opt)
 {
  wn_clr(w4);
  wn_puts(w4,2,5,"Enter option-- short phrase, please");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
  wn_locate(w2,p2_opt+11,3);
  wn_gets(w2,option,4,0);
  if (*option == NULL)
```

```
        break ;
    wn_putsa(w2,p2_opt+11,3,option,Blu_Wht);
    }
            wn_sync(w2,FALSE);
            v_hidec();
/*-------------------------------*/
/*   Generate allVectors          */
/*-------------------------------*/
            out_come:
                    num_opt = p1_opt + p2_opt;
                    for (i=0,limit=1; i<num_opt; ++i)
                            {
                                    limit = limit * 2;
                            }

                    wn_clr(w3);
                    wn_clr(w4);
                    sprintf(prn_buf,"%d",limit);
                    wn_puts(w4,2,5,prn_buf);
                    wn_puts(w4,2,10,"possible combinations ");
Generate(p1_opt,p2_opt,limit,Prefvec_1);
/*-------------------------------*/
/* Load a knowledge base into KES       */
/*-------------------------------*/
            if (exp_flg == TRUE)
             {
                    KES_ld_kb(KB_FILE, MEM_SIZE );
                    limit = exp_setup(Prefvec_1,limit,p1_opt);
             }
/*-------------------------------*/
/* Redisplay outcomes after removals  */
/*-------------------------------*/
            if (exp_flg == TRUE)
            {
                    New_out(p1_opt,p2_opt,limit,Prefvec_1);
            }
            wn_puts(w3,0,20,"Outcomes Remaining");
            wn_putsa(w3,0,50,player1,Red_Yel);
/*-------------------------------*/
/* copy prefvec 1 into prefvec 2     */
/*-------------------------------*/
            for (i=0; i<256; ++i)
            {
                    for (j=0; j<20; ++j)
                    {
                            Prefvec_2[j][i] = Prefvec_1[j][i];
                    }
            }
 wn_clr(w4);
 wn_puts(w4,2,10,"CR,to rank player 1-- h for help ");
if (silent_flg == FALSE)
            {
                    snd_confirm();
            }
```

```
 wn_locate(w4,2,45);
 wn_gets(w4,pnl_in,6,ok1_lst);
        if (*pnl_in == 'h')
                goto pop_pnl;
/*-------------------------------*/
/* Rank outcomes after removals by expert */
/*-------------------------------*/
        if (exp_flg == TRUE)
        {
                rank_out(Prefvec_1,limit,p1_opt);
        }
/*-------------------------------*/
/* Redisplay outcomes after removals  */
/*-------------------------------*/
        if (exp_flg == TRUE)
        {
                alter_vec(Prefvec_1,limit,player1,p1_opt,p2_opt);
        }
/*-------------------------------*/
/* Run stability on Prefvec_1       */
/*-------------------------------*/
        if (exp_flg == TRUE)
        {
                for (i=0; i<limit; ++i)
                {
                        stable(p1_opt+1,num_opt,i,Prefvec_1);
                }

   wn_clr(w4);
   wn_puts(w4,2,10,"CR,to rank player 2-- h for help ");
                if (silent_flg == FALSE)
                {
                        snd_confirm();
                }
   wn_locate(w4,2,45);
   wn_gets(w4,pnl_in,6,ok1_lst);
                if (*pnl_in == 'h')
                        goto pop_pnl;
}
/*-------------------------------*/
/* Rank outcomes after removals by expert */
/*-------------------------------*/
        if (exp_flg == TRUE)
        {
                rank_out(Prefvec_2,limit,p1_opt);
        }
/*-------------------------------*/
/* Redisplay outcomes after removals  */
/*-------------------------------*/
        if (exp_flg == TRUE)
        {
                alter_vec(Prefvec_2,limit,player2,p1_opt,p2_opt);
        }
/*-------------------------------*/
```

```
/* Run stability on Prefvec_2        */
/*-------------------------------*/
        if (exp_flg = = TRUE)
        {
                for (i=0; i<limit; + +i)
                {
                        stable(1,p1_opt,i,Prefvec_2);
                }
  wn_clr(w4);
  wn_puts(w4,2,10,"CR,to solve-- h for help ");
                if (silent_flg = = FALSE)
                {
                        snd_confirm();
                }
  wn_locate(w4,2,45);
  wn_gets(w4,pnl_in,6,ok1_lst);
                if (*pnl_in = = 'h')
                        goto pop_pnl;
        }
/*-------------------------------*/
/* Skip manual ordering if expert on   */
/*-------------------------------*/
        if (exp_flg = = TRUE)
        {
                len_vec1 = limit;
                len_vec2 = limit;
                KES_free_kb();
                goto busy;
        }
/*-------------------------------*/
/* Start the ordering of decimal vectors  for player 1 */
/*-------------------------------*/
   wn_clr(w4);
   wn_puts(w4,2,10,"CR,to continue-- h for help ");
        if (silent_flg = = FALSE)
        {
                snd_confirm();
        }
   wn_locate(w4,2,45);
   wn_gets(w4,pnl_in,6,ok1_lst);
                 if (*pnl_in = = 'h')
                                        goto pop_pnl;
play1_in:
        horz_pos = 0;
 num_opt = p1_opt + p2_opt;
        quit = FALSE;
        for (len_vec1=0; len_vec1 < 256; + +len_vec1)
 {
                horz_pos = horz_pos % 45;
                if (horz_pos = = 0)
                        {
                                wn_clr(w3);
                                wn_puts(w3,0,20,"Preference Vectors");
                                wn_putsa(w3,0,50,player1,Red_Yel);
```

```
                    }
  wn_clr(w4);
  wn_puts(w4,2,5,"Enter VECTOR or + for binary-- ");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
  wn_locate(w4,2,45);
  wn_gets(w4,dec_vec,2,0);
        switch (*dec_vec )
                {
                        case NULL:
                                quit = TRUE;
                                break;
                        case '-':
                                goto want_out;
                                break;
                        case '+':
                        con2dec(num_opt,p1_opt,horz_pos, len_vec1,Prefvec_1);
        sprintf(dec_vec, "%d", Prefvec_1[0][len_vec1]);
                        wn_putsa(w3,2,horz_pos,dec_vec,Blu_Wht);
                        stable(p1_opt+1,num_opt,len_vec1,Prefvec_1);
                                break;
                        default:
                        con2bin(dec_vec,num_opt,len_vec1,Prefvec_1);
                        wn_putsa(w3,2,horz_pos,dec_vec,Blu_Wht);
                        stable(p1_opt+1,num_opt,len_vec1,Prefvec_1);

                        for (i=1, space=3 ; i<=num_opt ; ++i)
                {
                if (i > p1_opt)
        space = 10 - p1_opt ;
        sprintf(dec_vec, "%d", Prefvec_1[i][len_vec1]);
        wn_puts(w3,i+space,horz_pos,dec_vec);
                }
                                break;
                }
                if (quit == TRUE)
                        break;
                horz_pos = horz_pos + 3;

 }
  wn_clr(w4);
  wn_puts(w4,2,10,"CR,to player 2-- h for help ");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
  wn_locate(w4,2,45);
  wn_gets(w4,pnl_in,6,ok1_lst);
                if (*pnl_in == 'h')
                                goto pop_pnl;
/*-------------------------*/
/* Start the ordering of decimal vectors for player 2 */
```

```
/*-------------------------*/
play2_in:
        wn_clr(w3);
 wn_puts(w3,0,20,"PREFERENCE VECTORS");
 wn_putsa(w3,0,50,player2,Red_Yel);
        horz_pos = 0;
        quit = FALSE;
 for (len_vec2=0; len_vec2 < 256; ++len_vec2)
  {
                horz_pos = horz_pos % 45;
                if (horz_pos == 0)
                        {
                                wn_clr(w3);
                                wn_puts(w3,0,20,"Preference Vectors");
                         wn_putsa(w3,0,50,player2,Red_Yel);
                        }
 wn_clr(w4);
 wn_puts(w4,2,5,"Enter VECTOR or + for binary");
        if (silent_flg == FALSE)
        {
                snd_confirm();
        }
 wn_locate(w4,2,45);
 wn_gets(w4,dec_vec,2,0);
         switch (*dec_vec )
                {
                        case NULL:
                                quit = TRUE;
                                break;
                        case '-':
                                goto want_out;
                                break;
                        case '+':
                        con2dec(num_opt,p1_opt,horz_pos, len_vec2,Prefvec_2);
        sprintf(dec_vec, "%d", Prefvec_2[0][len_vec2]);
                        wn_putsa(w3,9,horz_pos,dec_vec,Blu_Wht);
                        stable(p1_opt+1,num_opt,len_vec2,Prefvec_2);
                                break;
                        default:
                        con2bin(dec_vec,num_opt,len_vec2,Prefvec_2);
                        wn_putsa(w3,9,horz_pos,dec_vec,Blu_Wht);
                        stable(1,p1_opt,len_vec2,Prefvec_2);

                        for (i=1 ,space=3 ; i<=num_opt ; ++i)
                {
                if (i > p1_opt)
                space = 10 - p1_opt ;
                sprintf(dec_vec, "%d", Prefvec_2[i][len_vec2]);
                wn_puts(w3,i+space,horz_pos,dec_vec);
        }
                                break;
                        }
                if (quit == TRUE)
                        break;
```

```
                horz_pos = horz_pos + 3;

 }
   wn_clr(w4);
   wn_puts(w4,2,10,"CR,for UI s-- h for help ");
        if (silent_flg = = FALSE)
        {
                snd_confirm();
        }
   wn_locate(w4,2,45);
   wn_gets(w4,pnl_in,6,ok1_lst);
                 if (*pnl_in = = 'h')
                                        goto pop_pnl;
/*-------------------------------------*/
/*  Busy message to user                */
/*-------------------------------------*/
busy:
        wn_clr(w3);
        wn_putsa(w3,7,10,"BACK IN A MOMENT",Red_Yel);
        if (silent_flg = = FALSE)
        {
                snd_error();
        }
/*-------------------------------------*/
/*  Find the same vector in 2           */
/*-------------------------------------*/
 Find_same(Prefvec_1,Prefvec_2,same_vec,len_vec1,len_vec2);
/*-------------------------------------*/
/*  Solve the UI's into r,s,or u        */
/*-------------------------------------*/
 solve(len_vec1,len_vec2,Prefvec_1,Prefvec_2,ans1,ans2);
/*-------------------------------------*/
/*  Simultaneous Sanction Check         */
/*-------------------------------------*/
 simul_sanct(len_vec1,len_vec2,ans1,ans2,same_vec,Prefvec_1,Prefvec_2);
/*-------------------------------------*/
/*  Solve r,s,u into equilibriums       */
/*-------------------------------------*/
 equilibrium(len_vec1,ans1,ans2,Ans,Prefvec_1,Prefvec_2,same_vec);
/*-------------------------------------*/
/*  Final Stability table               */
/*-------------------------------------*/
        fin_table(len_vec1,len_vec2,ans1,ans2,Ans,Prefvec_1,Prefvec_2);
/*-------------------------------------*/
/*  Control panel operation             */
/*-------------------------------------*/
cntrl_pnl:
     wn_clr(w4);
     wn_puts(w4,2,10,"CR--QUIT or h--HELP");
     wn_locate(w4,2,45);
     wn_gets(w4,pnl_in,6,ok1_lst);
                                        pw1 = 0;
pop_pnl:
                                        if (*pnl_in = = 'h' )
```

```
    {
                                        pw1 = popup(0,5,5,33,14,Yel_Grn,Grn_Wht,&popinfo,TRUE) ;
                                switch (pw1)
                                {
                                        case 1:
                                                goto play1_in;
                                                break;
                                        case 2:
                                                goto play2_in;
                                                break;
                                        case 3:
                                                goto new_game;
                                                break;
                                        case 4:
                                                user_help();
                                                goto cntrl_pnl;
                                                break;
                                        case 5:
                                                diagnostic();
                                                if (user_clrflg = = TRUE)
                                                        disp_work();

                                                goto cntrl_pnl;
                                                break;
                                        case 6:
                                                ui_review(len_vec1,len_vec2,ans1,ans2,Ans,Prefvec_1,Prefv
ec_2);
                                                goto cntrl_pnl;
                                                break;
                                        case 7:
                                                exp_flg = TRUE;
                                                goto out_come;
                                                break;
                                        case 99:
                                                goto cntrl_pnl;
                                                break;
                                        default:
                                                goto want_out;
                                                break;
                                }
                        }
want_out:
        wn_close(w4);
        wn_close(w3);
        wn_close(w2);
        wn_close(w1);
        wn_restore(w0);
        if (silent_flg = = FALSE)
                {
                        laff();
                }
/*-------------------------------------------------*/
/* This is the last statement in Main and will automatically terminate */
/*-------------------------------------------------*/
```

```
}/* end  */
```

Bibliography

1. Andriole, Stephen J. "Artificial Intelligence Comes of Age," National Defense Journal, 69: 43-46 (December 1984).

2. Barr, Avron and Fiegenbaum, Edward A. The Handbook of Artificial Intelligence, Volumes 1&2. Los Altos, California: William Kaufman, Inc., 1981.

3. Fraser, Niall M. and Hipel, Keith W. Conflict Analysis Models and Resolutions, New York: North-Holland, 1984.

4. Harmon, Paul and King, David. Artificial Intelligence in Business Expert Systems, New York: John Wiley & Sons, Inc, 1985.

5. Hillier, Frederick S. and Lieberman, Gerald J. Introduction to Operations Research, 3rd edition. Oakland: Holden-Day, Inc, 1980.

6. Howard, N. Paradoxes of Rationality. Cambridge: MIT Press, 1971.

7. Kernighan, Brian W. and Ritchie, Dennis M. The C Programming Language. Englewood Cliffs: Prentice-Hall, Inc. 1978.

8. Orr, George E. Major, USAF. Combat Operations C^3I: Fundamentals and Interactions. Maxwell AFB: Air University Press, 1983.

9. Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill Company, 1983.

10. Schwartz, Richard I. and Shostak, Robert E. "The AI Challenge," PC Tech Journal, Vol. 4, No. 9: 195-197 (Sept 1986).

11. Skantze, Gen Lawerence A. "Project Forecast II-A Glimpse at Tomorrow's C3I," Signal: 32-33, (July 1986).

12. Staff of Software A&E. Knowledge BAse Author's Manual KES PS, Software Architecture and Engineering, Inc. 1986.

13. Staff. "USAF Lab Simulates Battle Management Tasks," Aviation Week & Space Technology, Vol 123, No. 23: 105 (Dec 1985).

14. Tzu, Sun. The Art of War, Translated by James Clavell, 1983.

15. User's Guide. Decisionmaker The Conflict Analysis Program. Waterloo,Ontario: Waterloo Engineering Software, (November 1986).

16. Von Neumann, J. and Morgenstern, O. Theory of Games and Economic Behavior, 3rd edition. Princeton: Princeton University Press, 1953.

17. Waterman, Donald A. A Guide to Expert Systems. Reading, Massachusetts: Addison Wesley, 1986.

18. Winston, Patrick Henry. Artificial Intelligence (Second Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1984.

19. Wohl, Joseph G. "Force Management Decision Requirements for Air Force Tactical Command and Control," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 9: 619-620 (Sep 1981).

20. Yanaros, John. "Conflict Analysis Final Project'" Report to Col O'Connell. Air Force Institute of Technology, Wright Paterson AFB, September 1986.

21. Yu-Cheng Liu and Gibson, Glenn A. Microcomputer Systems: The 8086/8088 Family Architecture, Programing, and Design, Second Edition. Englewood Cliffs: Prentice Hall, Inc. 1986.

VITA

PII Redacted

Major Rollin J Lutz, Jr. [redacted] He graduated from Carbondale High School in Carbondale, Illinois in 1968 and attended the University of Illinois from which he received the degree of Bachelor of Science in Physics in June 1972. Upon graduation, he enlisted in the Air Force and was commissioned through the Officer Training School in February of 1973.

After completion of pilot training at Laughlin AFB in 1974, and F-4 training at Homestead AFB he was assigned to Udorn RTAFB, Thailand. His remote tour in Thailand was completed 11 months later as the US withdrew its fighter aircraft from Southeast Asia, and he was reassigned to Clarke AFB, Phillipines. Returning to the US in 1975, he served as an airborn Forward Air Controller (FAC) in the Cessna O-2 providing close air support for Army units in the Eastern United States. Assignment to the A-10 attack aircraft came in 1978, as an acceptance pilot in Air Force Systems Command. Reassignment in the A-10 to Myrtle Beach AFB followed in 1982. At Myrtle Beach he served as an instructor pilot and flight commander until he entered Air Force Institute of Technology in 1985.

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release; Distribution Unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GST/ENS/87J	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENS	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) Rollin J Lutz, M.S., Maj, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____ 14. DATE OF REPORT (Year, Month, Day) 1987 June 15. PAGE COUNT 148	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Expert System, Conflict Analysis,
Artificial Intelligence

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: APPLICATION OF ARTIFICIAL INTELLIGENCE IN A CONFLICT ANALYSIS DECISION AID

Approved for public release: IAW AFR 190-1
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFIT)
Wright-Patterson AFB OH 45433

Thesis Chairman: Michael J. O'Connell, Ph.D., Col, USAF
Head, Dept of Operational Sciences

Gregory S. Parnell, Ph.D., Lt Col, USAF
Assistant Professor of Operations Research

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael J. O'Connell, Col, USAF	22b. TELEPHONE (Include Area Code) (513) 255-3362 22c. OFFICE SYMBOL AFIT/ENS

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Block 19:

This report documents the research conducted to develop a decision aid for an operations research methodology, conflict analysis. The purpose of this research was to demonstrate the feasibility of blending artificial intelligence and operations research techniques. A prototype system was designed, developed, and implemented in the form of an expert system. Unique in this design, was the expert system embedded completely within a conventional C language program. The prototype automated many of the tedious calculations and processes performed in a classic conflict analysis.

The development process was conducted in three phases. First, the system requirements were established. Second, design tradeoffs were made in the choice of tools and software. Third, the prototype was built using two parallel development tracks. Separate design components were completed in the conventional C language program and in the expert system program. When fully tested and debugged, the two programs were integrated within the C language program. Proper system performance was evaluated by comparison of results using a varied selection of classic problems. Finally, the results of the research are presented with recommendations for future work.